NASA Technical Memorandum 85832

Investigation of Fast Initialization of Spacecraft Bubble Memory Systems

Karen T. Looney, Charles D. Nichols, and Paul J. Hayes

June 1984

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

SUMMARY

Bubble domain technology offers significant improvement in reliability and functionality for spacecraft onboard memory applications. In considering potential memory system organizations, minimization of power in high capacity bubble memory systems necessitates the activation of only the desired portions of the memory. In power strobing arbitrary memory segments, a capability of fast turn-on is required. Bubble device architectures, which provide redundant loop coding in the bubble device, limit the initialization speed. Alternate initialization techniques have been investigated to overcome this design limitation. An initialization technique using a small amount of external storage has been demonstrated, using software written in 8085 assembly language and PL/M. This technique provides several orders of magnitude improvement over the normal initialization time.

INTRODUCTION

Bubble memory systems are quickly becoming a preferred storage medium in environments where a non-volatile storage medium is required. The utilization of a bubble storage system offers the benefits of increased reliability, reduced maintainance, and permanent data integrity. The implementaton of large bubble memory systems in spacecraft applications requires that the memory modules be power strobed for the conservation of the available energy resources. Each time a module is turned on for use it must be initialized to code the redundant loop information of the selected bubble devices into the bubble controller. Present structures of bubble systems dictate that a faster initialization procedure is needed in order to capitalize on the advantages offered by a bubble memory system. Use of brand or trade names herein does not imply NASA endorsement.

NEED FOR FAST INITIALIZATION

A proposed structure for a large spacecraft memory system(ref. 1) is shown in figure 1. The system controller module translates high level user commands into simple digital signals for use by the bubble controller. The bubble controller takes these digital signals and outputs the specific current and voltage levels and timing characteristics that are required to drive the bubble memory boards. Each of these memory boards contains N parallel devices, where N is determined by the desired system data rate. The size of the overall system is determined by the number of memory boards present. Since each of these boards require about 50 watts of power, they cannot be left powered up in a spacecraft environment where power is limited. It is more advantageous to power up a single board only when data needs to be read from or written to that particular section of memory. Fast access to a portion of memory now becomes a function of how quickly a board can be turned on. Current bubble device architecture limits the amount of time it takes to initialize a memory device to prepare it for use.

The functional organization of all bubble devices used today is a major track/minor loop architecture similar to that shown in figure 2. Refer to Appendix A for a more detailed description of bubble memory devices. This architecture is desirable because it provides a shorter access time than the previous serial designs and improves manufacturing yields in high density devices. Each minor loop is for the storage of data, while the major tracks provide the input and output circuitry. Up to 15% of these loops are redundant to allow for processing defects. The location of the defects will

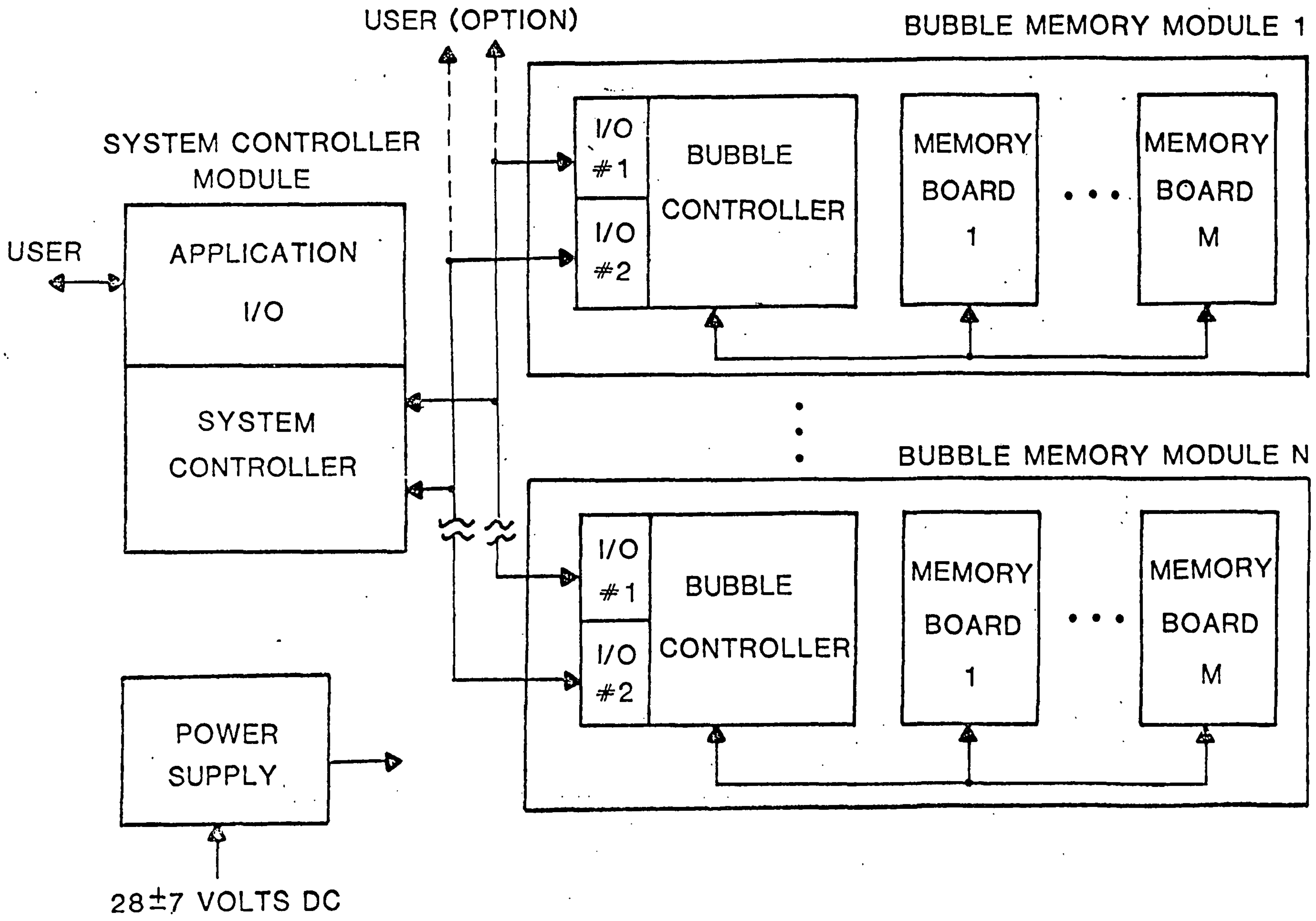
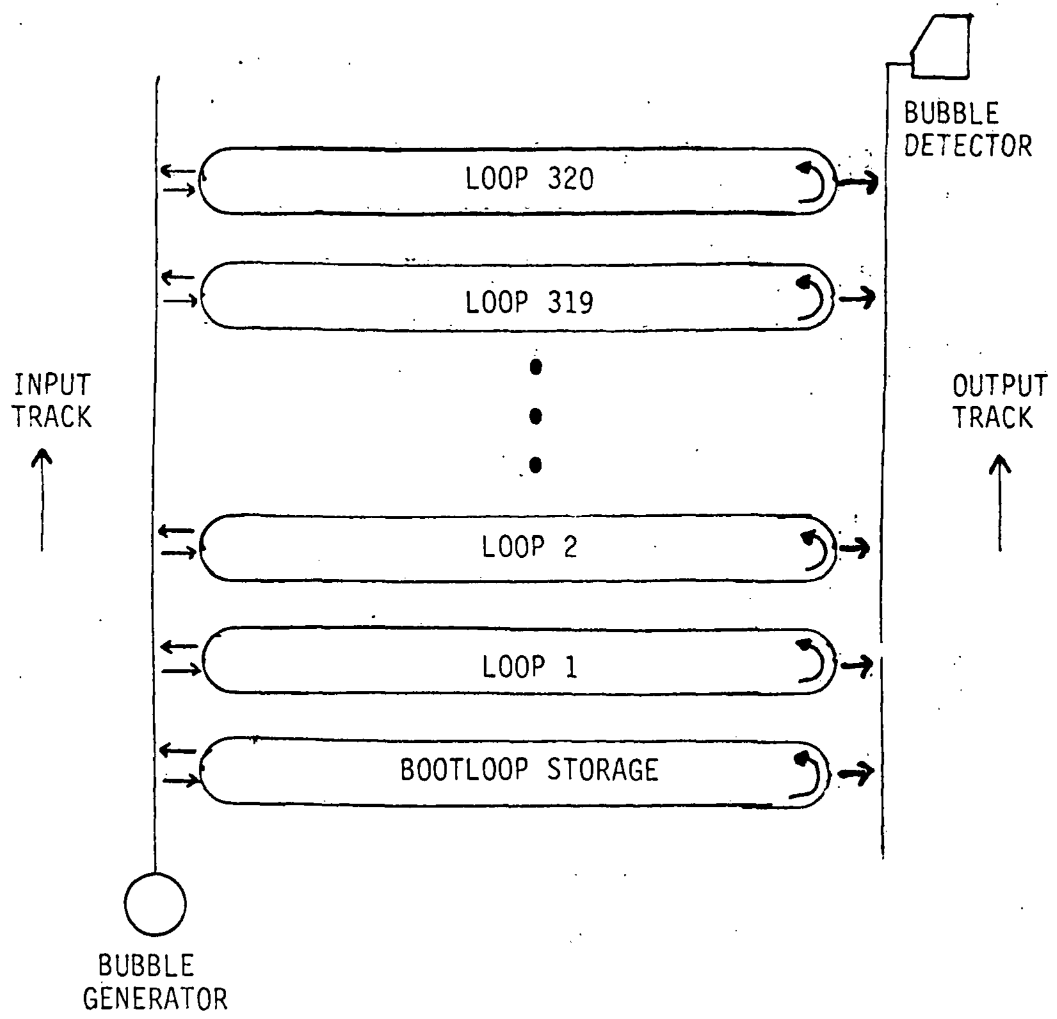


FIGURE 1.- BUBBLE MEMORY SYSTEM BLOCK DIAGRAM (REF. 1)



- 4096 BITS PER LOOP
- 15% REDUNDANT LOOPS

FIGURE 2.- FUNCTIONAL ORGANIZATION OF A BUBBLE CHIP (REF. 2)

vary from device to device, but post-fabrication testing can determine which loops are available for use. The code used to identify the good loops is currently stored in a separately accessible bootloop on the device. Initialization of most commercially available bubble devices requires the readout of the entire bootloop to a register in the control circuitry. The time required for this operation is determined by the rotation time of the field coils and the length of the minor loop. In application this initialization scheme could result in a loss of data because the memory may not be ready to accept data when the data is ready to be transmitted.

The achievement of higher data rates requires the paralleling of the bubble devices. Table 1 shows the number of devices required to produce a desired average useful data rate and the initialization times needed for each configuration using the Intel 1 Mbit bubble device. These initialization times are a reasonable estimation for other manufacturers devices. Intel's device was chosen to demonstrate the fast initialization concept due to its commercial availability. The architecture of the Intel controller (BMC) dictates that each bubble device be initialized in series, even though the devices can operate in parallel. The initialization time required for a 1 megabit data rate is on the order of 2 seconds, which is a very high time price to pay every time a device needs to be accessed. It is more desirable to get the initialization time down into the millisecond or microsecond range so that a memory segment can be accessed quickly to accommodate complicated mission scenarios. Since the present initialization scheme is dependent on the internal architecture of the device, the bootloop data must be stored externally in order to provide a faster initialization.

Table 1. Initialization Times

#OF DEVICES	AVERAGE USEFUL DATA RATE	INITIALIZATION TIME	
		NORMAL	MAXIMUM
1	68 kb/s	80 ms	160 ms
8	544 kb/s	640 ms	1280 ms
16	1088 kb/s	1280 ms	2560 ms

FAST INITIALIZATION

Specific Implementation

A single Intel bubble memory device was used to test the concept of fast initialization. The result of this experiment can be applied to higher density systems and to other manufacturers' devices. Figures 3 and 4 show the flow charts for the normal and the fast initialization procedures. The main difference between the normal and fast initialization procedures is a reliance on either hardware or software, respectively, for the completion of the initialization process. Both initialization procedures require a system reset (ABORT) prior to the actual command being sent to the controller. When the system is initialized normally, the bubble memory controller (BMC) reads the entire bootloop, decodes it, transfers it to the bootloop register in the format/sense amplifier (FSA), and places the bubble at page zero. This process could take up to 160 ms since there could be two rotations of the minor loops to find and read the bootloop code. Before calling the Intel initialization routine, the parametric registers must be properly set up in the RAM. The routine BMINIT in Appendix C was written to set up the 8085 registers and addressable memory. Alternatively, the fast initialization procedure uses system software to load the bootloop information from an external memory into the bootloop register of the FSA. The first different

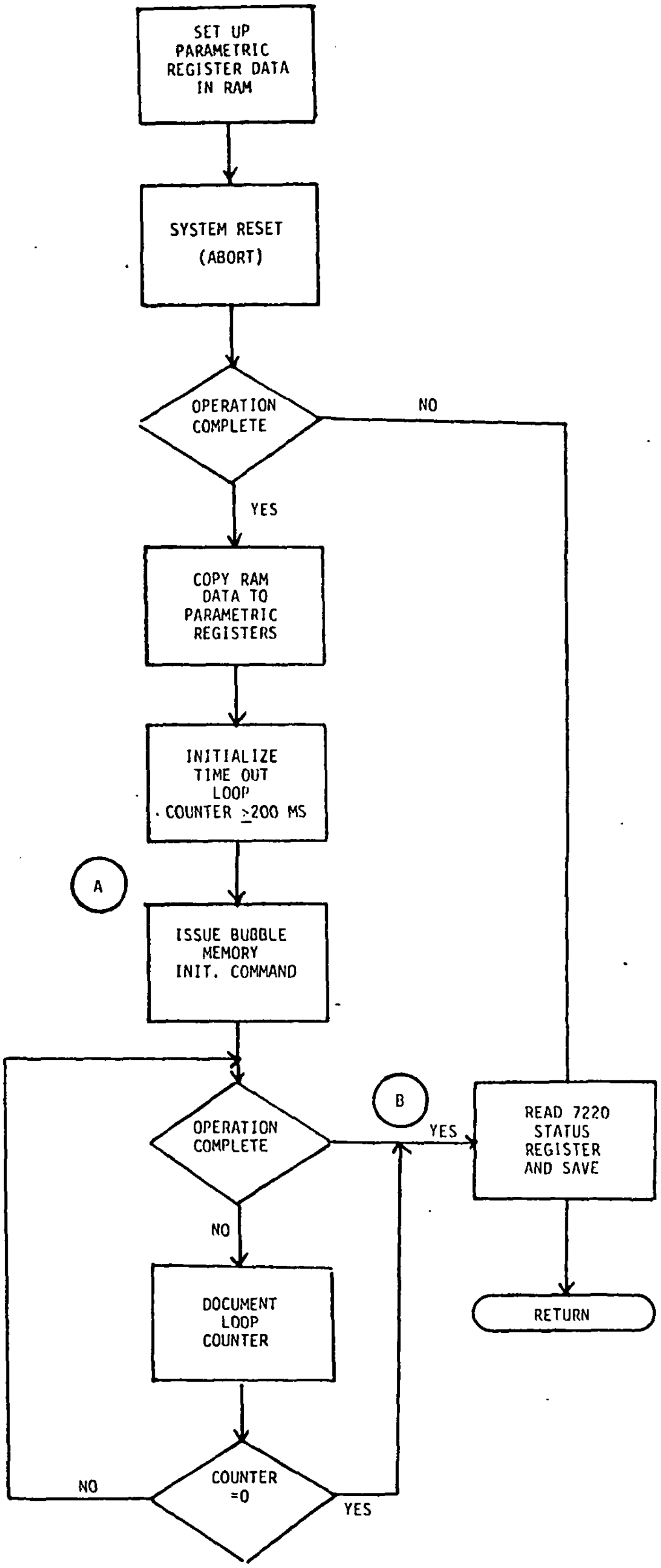

SET UP PARAMETRIC REGISTER DATA IN RAM
SYSTEM RESET (ABORT)
OPERATION COMPLETE
NO
YES
COPY RAM DATA TO PARAMETRIC REGISTERS
INITIALIZE TIME OUT LOOP COUNTER ≥200 MS
A
ISSUE BUBBLE MEMORY INIT. COMMAND
OPERATION COMPLETE
B
YES
READ 7220 STATUS REGISTER AND SAVE
NO
DOCUMENT LOOP COUNTER
RETURN
NO
COUNTER =0
YES

FIGURE 3.- FLOW CHART FOR NORMAL INITIALIZATION.

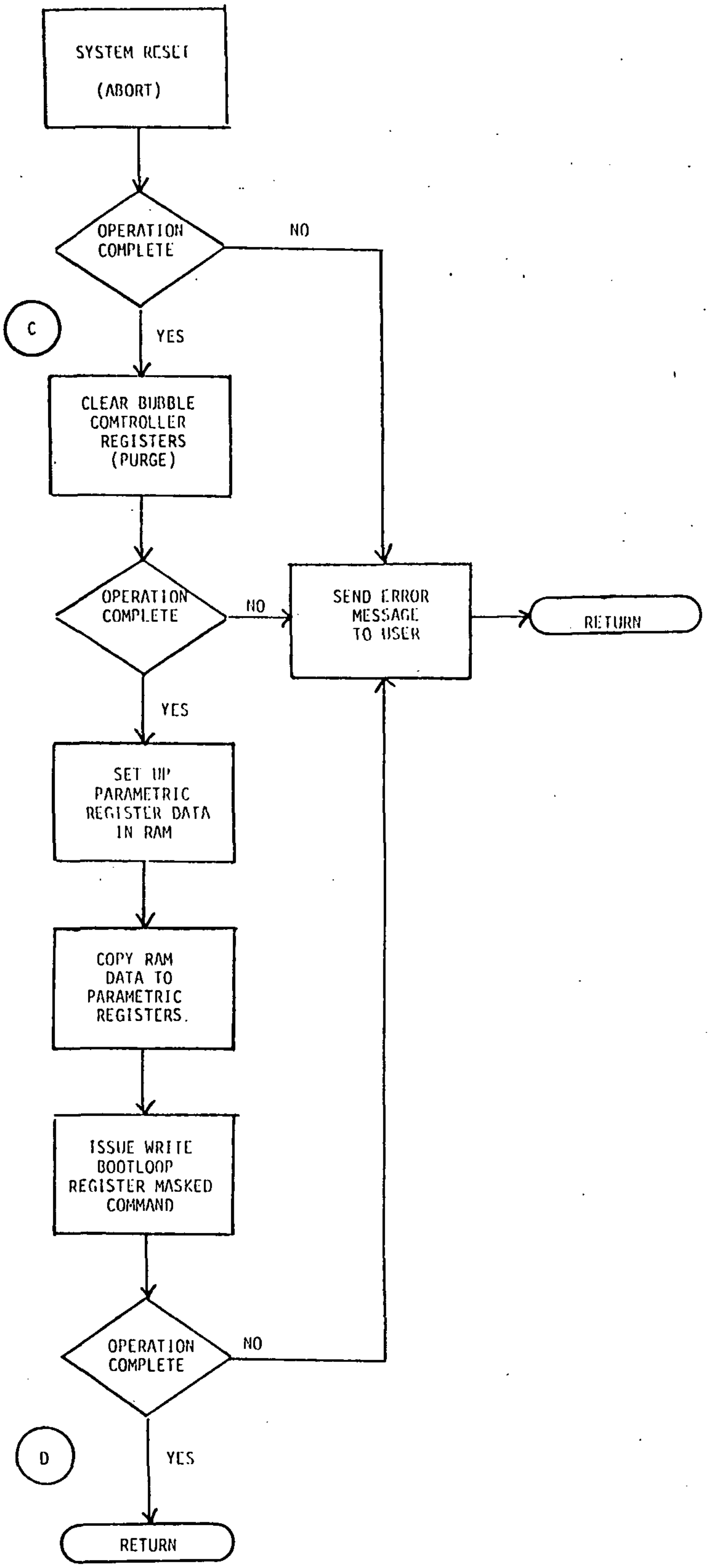

SYSTEM RESET
(ABORT)
OPERATION COMPLETE
NO
YES
C
CLEAR BUBBLE COMTROLLER REGISTERS (PURGE)
OPERATION COMPLETE
NO
SEND ERROR MESSAGE TO USER
RETURN
YES
SET UP PARAMETRIC REGISTER DATA IN RAM
COPY RAM DATA TO PARAMETRIC REGISTERS.
ISSUE WRITE BOOTLOOP REGISTER MASKED COMMAND
OPERATION COMPLETE
NO
D
YES
RETURN

FIGURE 4.- FLOW CHART FOR FAST INITIALIZATION.

step in the fast initialization process is to issue the purge command to clear the controller registers. Next the parametric register data must be loaded with the same data as required for the normal initialization. Upon completion of this, the bootloop register must be loaded from the external memory to the bootloop register in the FSA. This is accomplished by using the "write bootloop register masked" command, which insures the loading of exactly 272 ones in the bootloop register. (It is important to note that there must be *exactly* 272 ones in order for the read command to be properly implemented.) The fast initialization routine (FSTINT) in Appendix C implements this procedure. Table 2 contains a description of the alternative external storage methods that were considered. EPROM was chosen for this implementation because it was the simplest method available for the illustration of the concept. The external EPROM for this experiment was located on the single board computer, along with the system software. This configuration allowed for easy comparison of the normal and fast initialization processes.

Table 2. Alternate External Storage Methods

MEDIUM	ADVANTAGE	DISADVANTAGE
CORE	Rewrite capability in system Space qualified	Low density
ROM	High density Space qualified	Not Reprogrammable
EPROM	High density Reprogrammable Space qualified	No rewrite in system
E^2Prom	High density Rewrite capability in system	Required more extensive development

HARDWARE INTERFACE

The Intel bubble memory requires a smart controller to take commands from the user and translate them into digital commands for the bubble controller. The controller chosen for this experiment was an Intel 8085 based single board computer, the iSBC 8024. Interfacing the bubble memory to the computer was simply a matter of constructing an interface to the Intel multibus. A block diagram of the experimental configuration is shown in figure 5. Table 3 shows the necessary hardware interface signals and a schematic of the hardware interface can be found in Appendix B. The interface required the use of address buffers and decoding, data and control signal buffers, acknowledge decode and a clock generation circuit(ref. 3). Buffering of all signals that cross the bus is necessary so that there is no confusion about who has control of the bus and to prevent the garbling of data. Address decoding is required for selection of the bubble controller for issuing commands or the transmission of data. An acknowledge is needed from the bubble contoller to let the single board computer know the information has been received. The clock generation circuit must provide a 4 MHz asynchronous TTL level clock, according to the specifications given in Table 3. These clock tolerances must be strictly observed to assure the stability of the rotating field. More detail about microprocessor interface requirements can be found in Intel's AP-119(ref. 3). Our particular implementation required that the user be able to issue commands to the bubble memory module and receive status information back from the controller during the testing of the device. This required the connection of a terminal to the single board computer card in order for the user to control the system. It is important to note that DACK/ and WAIT/

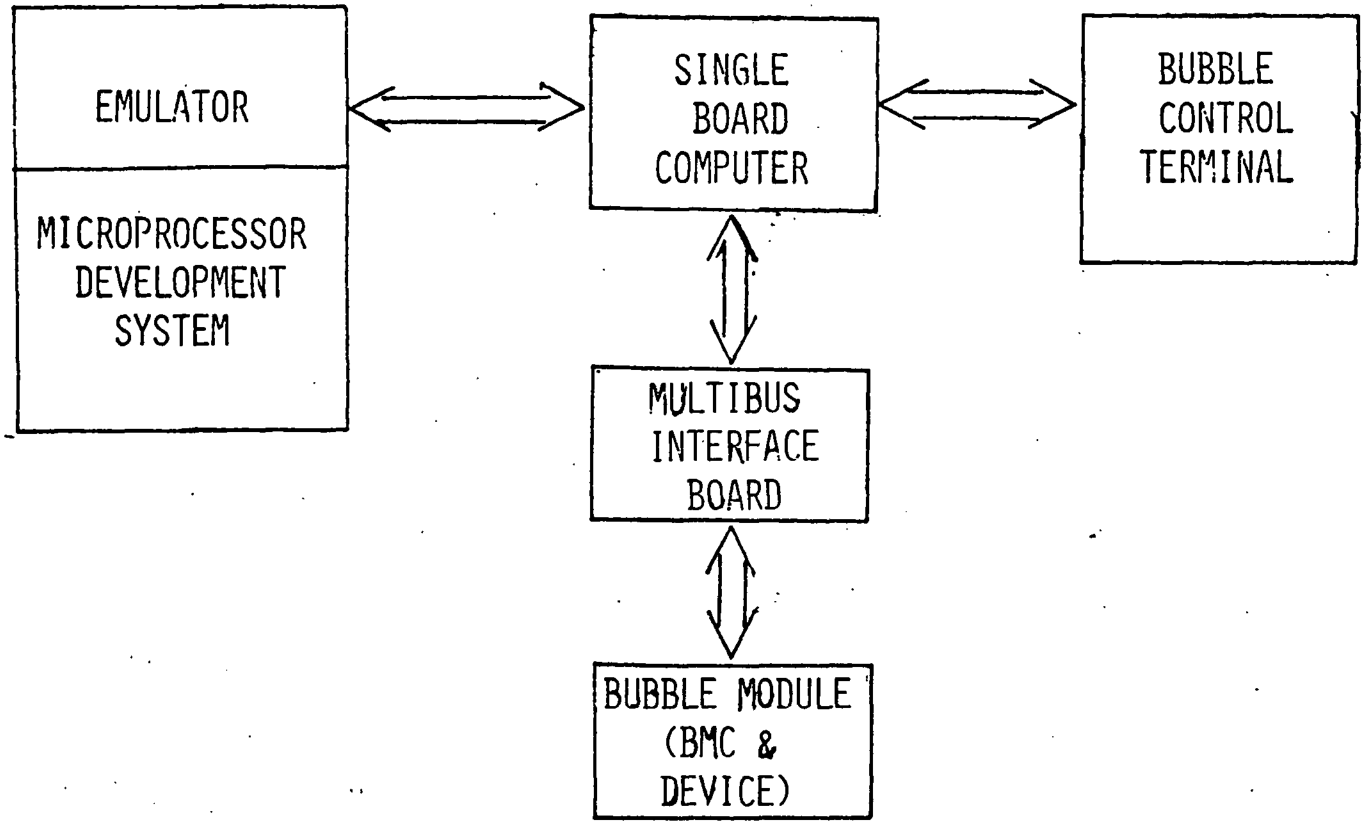


FIGURE 5.- EXPERIMENTAL CONFIGURATION.

should be tied to +5V, as shown in the diagram in Appendix B, or the controller will function erratically. Everything possible must be done to insure that the system is free of spurious signals on the data and control lines: decoupling capacitors should be used on all integrated circuits, all line lengths should be as short as possible, and the data lines should be in twisted pairs. The system will not function properly if any spurious signal activity is present.

Table 3. Bubble Device Interface Signals(ref. 4)

SIGNAL	FUNCTION

A0	Address line, selects:
	A0=0 FIFO or parametric registers
	A1=1 Command/Status registers

D0-D7	Bidirectional data bus
D8	Optional odd parity bit
7220 CS/	BMC chip select input CS/=0 controller select CS/=1 tri-state interface signals
RD/	Read 7220 registers or data FIFO
WR/	Write 7220 registers or data FIFO
DACK/	DMA acknowledge--requires an external pullup resister to V_{cc}(5.1 kohm)
WAIT/	Used when BMC's are operated in parallel--causes halt when an error is detected in a BMC--requires
CLK	4 MHz T^2L level clock clock period=250 ns(0.25 ns tolerance) duty cycle= 50% (5% tolerance)
RESET/	A low on this pin forces the interruption of any 7220 activity, performs a controlled shutdown and initiates a reset sequence
7242 CS/	Should be tied to ground for single bank systems

SOFTWARE INTERFACE

Intel Software Routines

The basic 8085 to BPK-72 (Intel's bubble module kit) software driver routines can be found in reference 4, along with a detailed description of each routine. These programs are a set of subroutines that can be called to perform commonly used bubble memory commands. This software driver is written in 8085 assembly language and can easily be incorporated into existing systems as part of a utility program to transfer data between a user and the bubble memory module. Usage of these driver routines requires that certain 8085 registers and specific locations in memory be properly set up in order to satisfy the restrictions placed on the user addressable registers. Additional software was also required for our application to take the commands from the user terminal and interpret them for the BMC. The software listed in Appendix C was written, in 8085 assembly language and PL/M-80, to utilize the Intel driver routines, get information to and from the terminal and perform the fast initialization command.

Table 4. Parametric Register Set Up for Initialization(ref. 4)

REGISTER	VALUE
Utility	anything
Block Length	1001H
Enable	00H
Address	0000H

Test Software

The test driver routines listed in Appendix C are the routines that were necessary to utilize the basic Intel driver routines. These routines form the interface between the user and the BMC. The main controlling program, BMCOM, continually takes the commands entered by the user and allows them to be executed if they are valid commands. The module DOCMD contains the programs necessary to actually implement the commands. Two modules, called BMIO and IOMOD, contain the programs which query the user about various transfer parameters (number of pages to transfer, once or continuously, page number desired in the bubble memory) and translate the requests for use by the BMC. The program modules used for getting messages to and from the terminal are: MENU, CHKVAL, CNVERT, ERRMOD, TERMIO, TSTMOD. All of these test driver modules are linked together with the Intel driver routines and an operating system for the iSBC 8024 board to make up the software package used to fully exercise the Intel bubble memory module.

TEST PROCEDURE

Comparison of the normal and fast initialization times was done by calculating the number of clock cycles (T-states) required by the microprocessor to complete the process(ref. 5). Each initialization routine was executed several times, with one bubble memory device in place, to determine the number of times any software loops within the routines were called. All of this data was used to calculate the time required to complete the normal and fast initialization processes.

TEST RESULTS

Calculation of the normal initialization time was done from the time the command was issued to the BMC (letter A in Fig. 3) to the time that an op complete status was received in the status register(letter B in Fig. 3). The loop POLLIN, within the program INBUBL, contains 61 T-states for each complete run through this loop. Twenty-seven T-states are required the last time through the loop when the op complete status is detected in the status register. A total of 9288 times through the loop was required to successfully complete the normal initialization process. With the 4.8 MHz clock that is used on the 8024 single board computer, the calculation for the normal initialization time (t_n) yields:

$$t_n = [(9287)61 + 27] \times (0.208 \text{ us/T-state}) = 117.84 \text{ ms}$$

This initialization time is for one bubble memory device. The time required to initialize 8 or 16 devices in parallel would be 8 or 16 times greater than the above calculated value. The results of these calculation are shown in Table 5.

Execution of the fast initialization process requires more software than the normal initialization process so the calculation of the time required to complete the process is slightly more complicated. Routines which are called by both the normal and fast initialization procedures were excluded from the calculations for purposes of comparing only the time required for the transference of the bootloop data to the controller. The calculations of the fast initialization time began with the issuance of the PURGE command (letter C in Fig. 4) and ended with the receipt of an op complete in the BMC status register after the completion of the "write bootloop register masked" (WRBLRM) command (letter D in Fig. 4). Calculation of the fast initialization time (t_f) is as follows:

$$\begin{aligned}\text{Total \# of T-states} &= (\text{PURGE})+(\text{CKSTAT})+(\text{error check})+(\text{WRBLRM})\\ &= 875 + 44 + 27 + 2136\\ &= 3082\end{aligned}$$

$$t_f = (3082) \times (0.208 \text{ us/t-state}) = 0.64106 \text{ ms}$$

The calculation for the initialization of 8 bubble devices in parallel is similar to the calculation above, with the exception that:

$$\text{WRBLRM}=368+(8)\text{WRFIFO} = 368+8(1768)$$

The "write FIFO" (WRFIFO) routine must be called to transfer the bootloop data for each bubble device. Resulting initialization times for 8 and 16 parallel bubble devices are shown in Table 5.

Table 5. Experimental Initialization Times(ms)

	# of parallel bubble devices		
	1	8	16
t_n (ms)	118.03	943.72	1885.44
t_f (ms)	0.641	3.215	6.43

The results of this experiment show that the fast initialization concept provides several orders of magnitude improvement over the normal initialization scheme. Although these results were obtained with an Intel 1 Mbit device, they are applicable to higher density devices. The improvement in the initialization time can be projected for similarly organized bubble devices of various capacities. For normal initialization

$$t_n \alpha \frac{\text{minor loop length}}{\text{field rotation rate}}$$

for fast initialization

$$t_f \; \alpha \;\; \text{minor loop length}$$

Several hypothetical device configurations which have been considered are listed in Table 6. The initialization times for these device configurations is depicted in figure 6 for a parallel group of 8 devices. The shaded region indicates the time range that is achievable for the fast initialization scenario, depending on the controlling microprocessor and the efficiency of the system software. The 50 kHz and 100 kHz lines represent the normal initialization times required for various device configurations. It is seen that very long initialization times can occur if fast initialization techniques are not incorporated. The lower limit of the shaded region is due to the minimum time to transfer data out of the EPROM.

Table 6. Bubble Device Configurations

Device	Capacity (bits)	Field Rate	# of Minor Loops	Minor Loop Length
A	256k	100k	256	1024
A'	1M	50k	256	1024
B	1M	100k	512	2048
B'	1M	50k	512	2048
Intel	1M	50k	272	4096
C	2M	100k	163	4103
D	8M	100k	587	4103
E	4M	50k	512	8192
E'	4M	100k	512	8192
F	4M	100k	163	8206

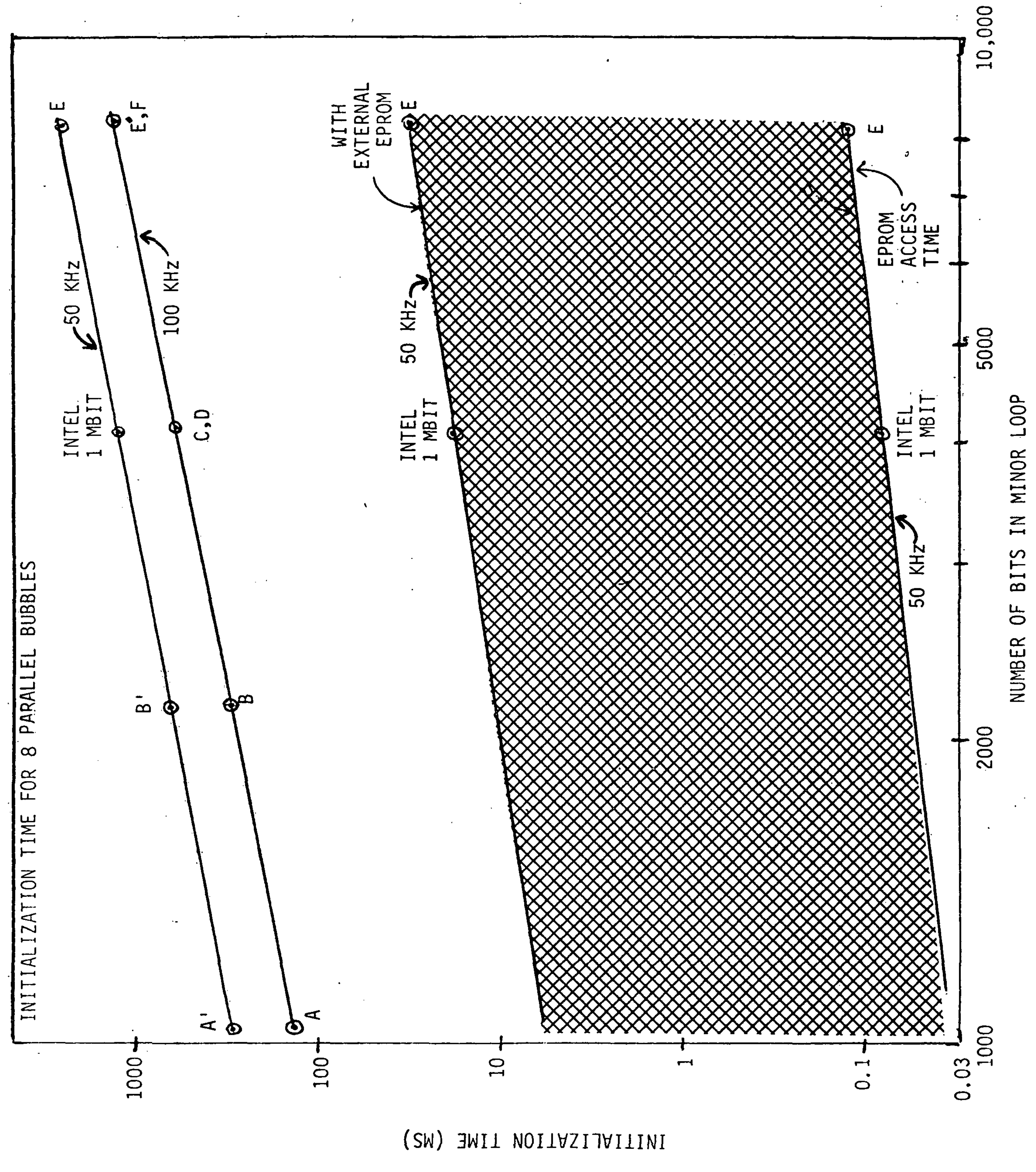


FIGURE 6.- GRAPH OF INITIALIZATION TIMES.

CONCLUDING REMARKS

A method for the fast initialization of a bubble memory system, involving the usage of a small amount of external storage, has been presented. Specific information on the implementation of an Intel 1 Mbit bubble memory device has also been included to illustrate the methodology. The fast initialization technique presented here is conceptually applicable to all bubble devices and to higher density systems.

After an evaluation of the external storage mediums available, EPROM was chosen as the simplest method to demonstrate the viability of this fast initialization technique with Intel devices. A hardware interface was designed to interface the controlling microprocessor to the bubble memory circuitry. System software was written to exercise the various functions of the bubble memory system. A comparison was made between the normal and fast initialization techniques. The fast initialization method has been demonstrated to reduce the initialization time by several orders of magnitude. Future implementations of this approach will utilize the E^2PROM to provide greater system flexibility.

REFERENCES

1. Hayes, P. J.; Stermer, R. L., Jr.: "Bubble Domain Technology for Spacecraft Onboard Memory," Advanced Remote Sensing: Proceedings of SPIE 26th Internat'l Technological Symposium, Vol. 363, pp. 136-146, Aug. 22-27, 1982.

2. Intel Corp.: "Bubble Memory System Design Workshop: Student Study Guide," Version 1.0, Nov. 1981.

3. Intel Corp.: "Microprocessor Interface for the BPK-72," Intel AP-119, June 1981.

4. Intel Corp.: "8085 to BPK-72 Interface," Intel AP-150, July 1982.

5. Intel Corp.: MCS-80/85 Family User's Guide, Oct.1979.

APPENDIX A

INTRODUCTION TO BUBBLE MEMORY DEVICES

Paul J. Hayes
Langley Research Center
May 1983

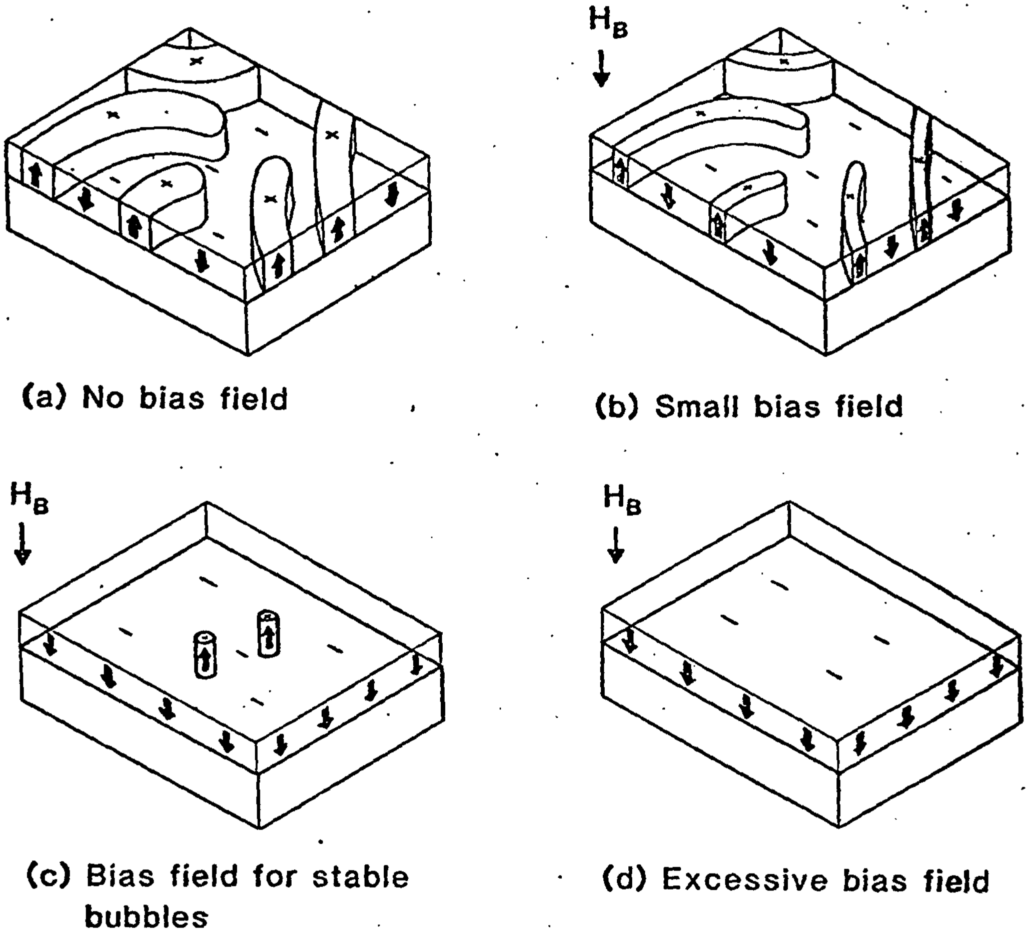
APPENDIX A

INTRODUCTION TO BUBBLE MEMORY DEVICES

Magnetic bubble memory chips are fabricated by depositing a thin magnetic garnet film on top of a non-magnetic garnet substrate. As sketched in figure 7(a), a random arrangement of serpentine magnetic domains forms in the magnetic film with half of the domains oriented in one direction and half in the opposite direction. The film deposition process induces anisotrophy in the magnetic film which causes the domains to be oriented perpendicular to the surface.

If a d.c. bias field H_B (external magnetic field) is now applied perpendicular to the surface, these serpentine domains already oriented in the direction of the bias field will grow at the expense of those oppositely oriented as depicted in figure 7(b). As the bias field strength is increased this process continues until a field value is reached where the oppositely-oriented serpentine domains shrink into small cylindrical domains (Fig. 7(c)). These cylindrically-shaped domains are referred to as bubble domains, or simply as bubbles, and are stable in their cylindrical shape over a reasonably wide range of magnetic bias field. Should the bias field be increased beyond the stable range the bubbles will finally collapse (reverse orientation to align with the bias field), resulting in a single domain aligned in the direction of the applied field (Fig. 7(d)).

The stable bias field for bubbles may be provided by a permanent magnet, thus maintaining a digital storage medium without applied power (nonvolatile). Digital data may be represented by the presence and absence of bubbles. The presence of a bubble may represent a logical "1" and the absence of a bubble may represent a logical "0." The condition for having stable



(a) No bias field

(b) Small bias field

(c) Bias field for stable bubbles

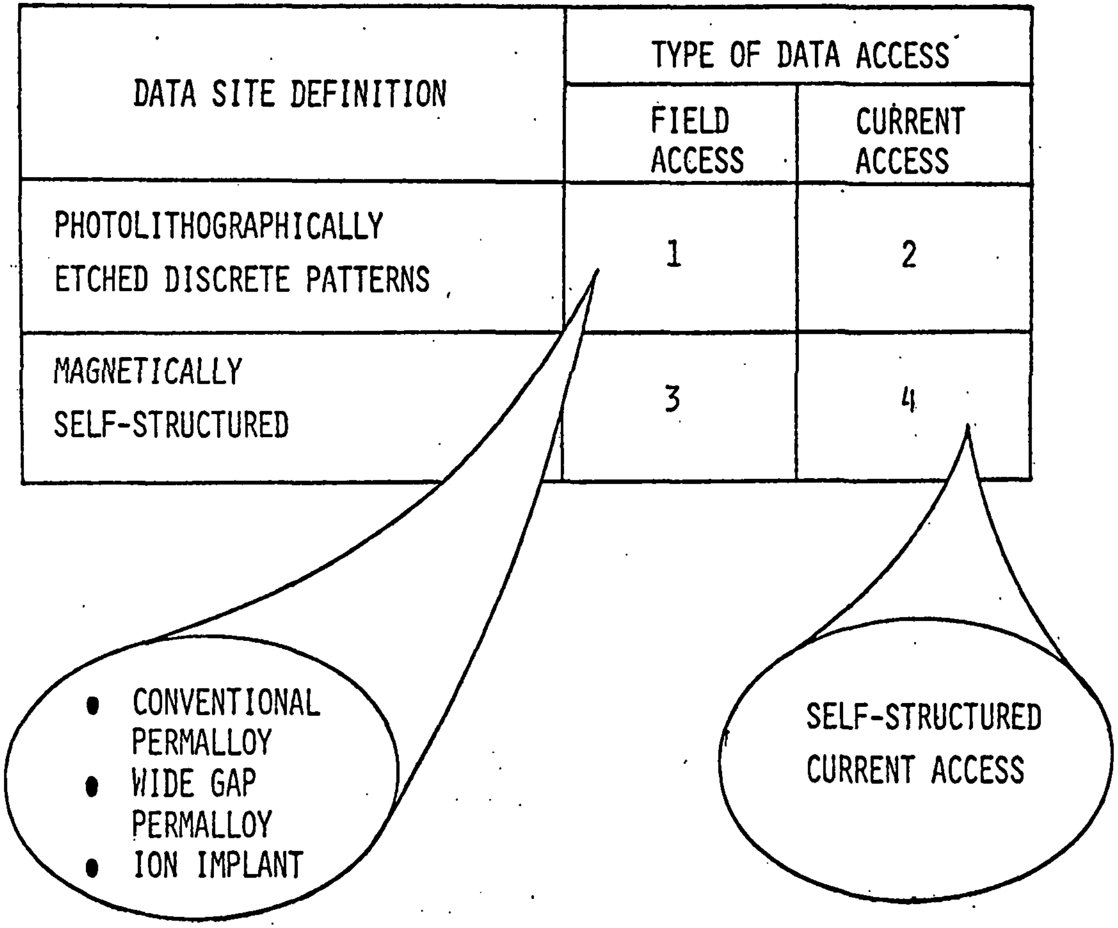
(d) Excessive bias field

Figure 7.- Domain patterns in magnetic garnet film.

bubbles is not by itself sufficient for practical application. Device features must be included which identify specific data storage sites, provide bubble generation, annihilation (erase), and detection techniques, and provide for propagation of bubbles to and from the storage sites.

Bubble devices may be classified by the technique used to define data positions and the method of propagating or accessing data. The four major classifications are indicated in figure 8. Three device types within classification 1 and one within classification 4 are currently being assessed for aerospace onboard applications. Two device types within classification 1, conventional permalloy and wide-gap permalloy, have matured sufficiently to be considered for near-term onboard system development. Figure 9 illustrates the shape of permalloy features identifying storage sites in these two device types. Rows of these asymmetric chevrons are fabricated over the top of the magnetic garnet film. Each permalloy chevron is a data storage site and the bubble is located in the garnet film underneath the chevron. The data site period determines the device density (density is inversely proportional to the square of the data site period). Conventional permalloy is further along in commercial applications but the wide-gap structure, by virtue of its shape and a 50% wider gap between sites, offers lower power, higher density, and potentially lower cost.

The propagation of data is now discussed for the conventional permalloy device, however, propagation is similar in the wide-gap device. The propagate structure (and also detectors, generators, etc.) is patterned over the magnetic garnet film as shown in figure 10(b) and the resulting chip is



DATA SITE DEFINITION	TYPE OF DATA ACCESS	
	FIELD ACCESS	CURRENT ACCESS
PHOTOLITHOGRAPHICALLY ETCHED DISCRETE PATTERNS	1	2
MAGNETICALLY SELF-STRUCTURED	3	4

- CONVENTIONAL PERMALLOY
- WIDE GAP PERMALLOY
- ION IMPLANT

SELF-STRUCTURED CURRENT ACCESS

Figure 8.- Four major classifications of bubble devices.

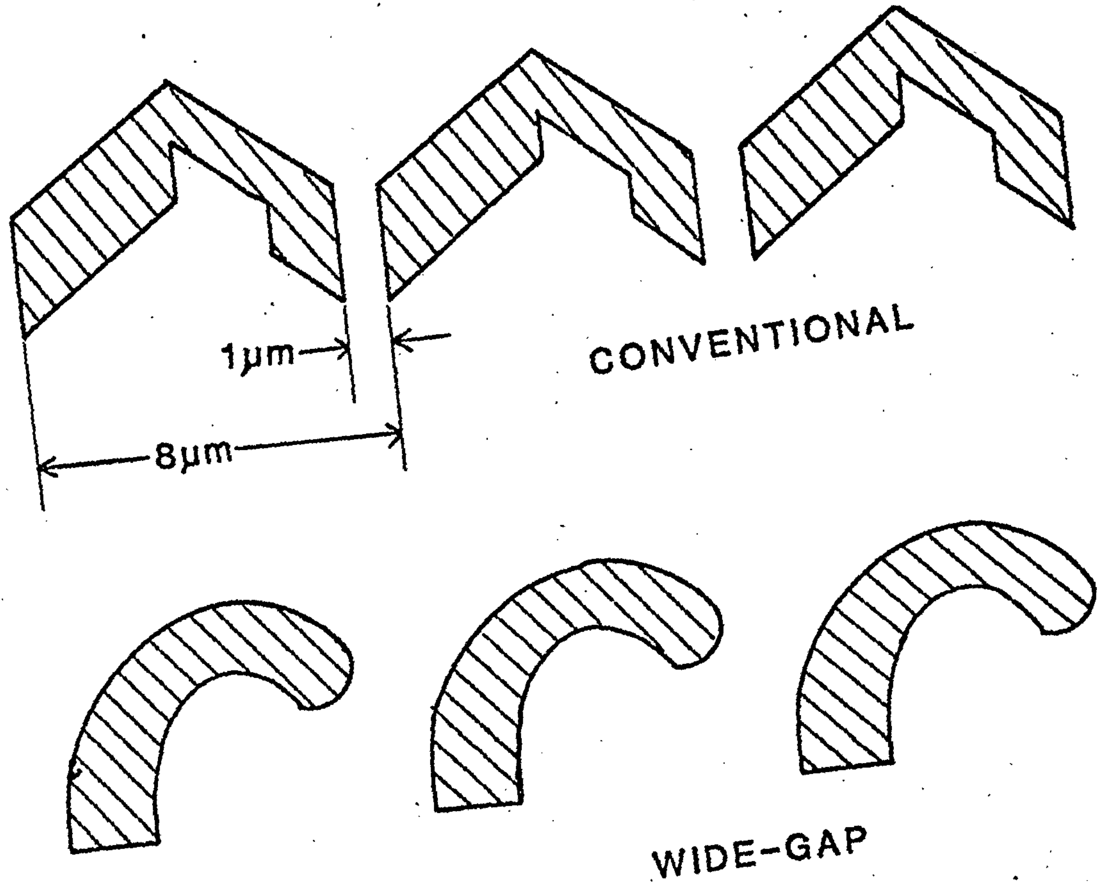
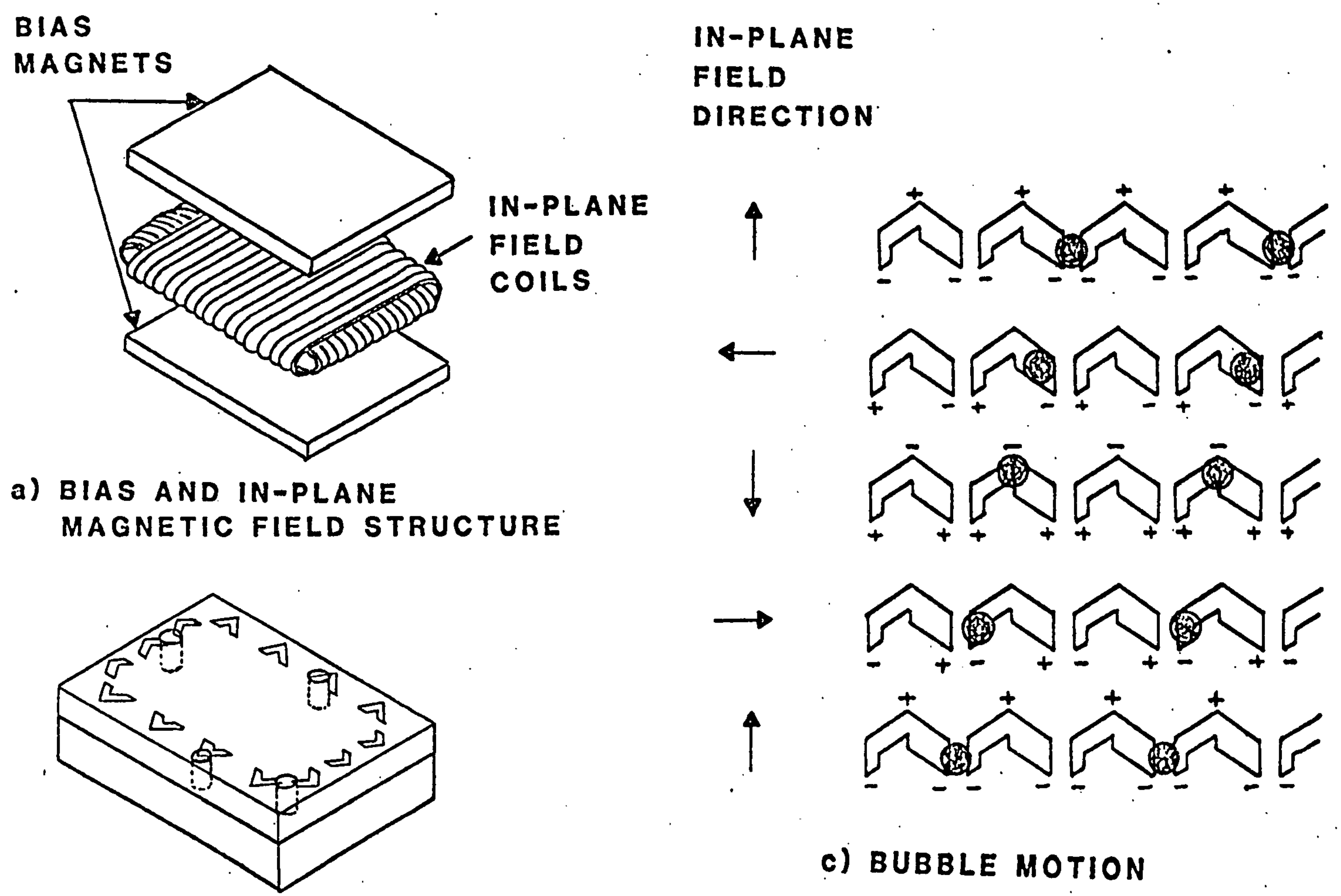


Figure 9.- Permalloy gap device patterns.



a) BIAS AND IN-PLANE MAGNETIC FIELD STRUCTURE

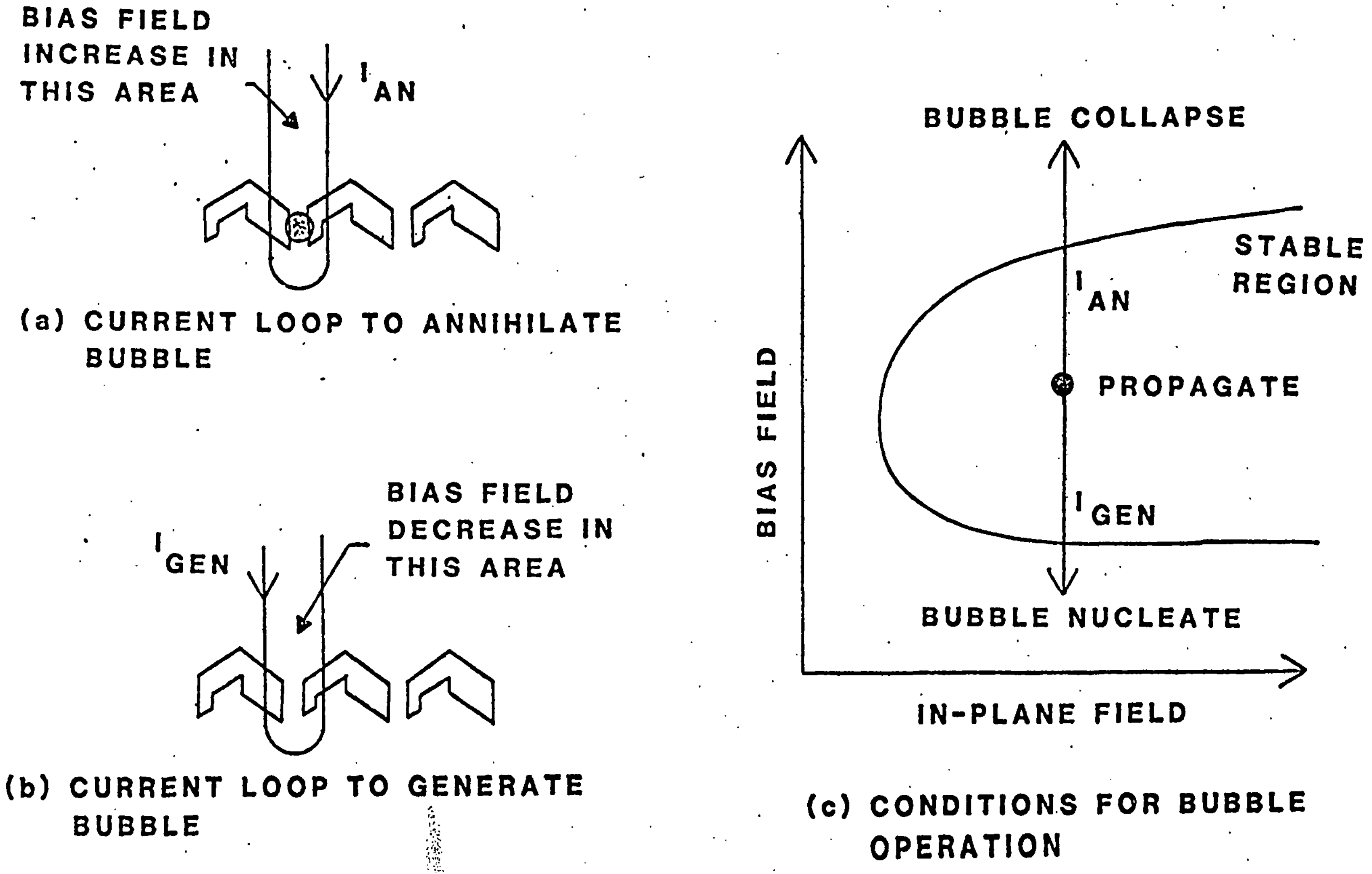
b) PROPAGATE STRUCTURE

c) BUBBLE MOTION

Figure 10,- Bubble propagation technique in conventional permalloy technology.

located inside a pair of orthogonal coils and between permanent magnets (Fig. 10(a)). The pair of coils is used to generate a rotating in-plane magnetic field. The magnetization of the permalloy chevrons aligns with the in-plane field so that each chevron simulates a small bar magnet which generates a local field to position the bubble. The bubbles are attracted to the magnetized chevrons and move along from one chevron to another as the in-plane field rotates (Fig. 10(c)). Bubble domain motion does not involve the moving of matter, but rather consists of reorienting the magnetizaion of adjacent microscopic regions of the garnet. This reorientation occurs very rapidly to enable fast propagation times and, hence, fast data rates.

Figure 11 illustrates the conditions for bubble operation and techniques for bubble generation and erasure. The nominal operating point (bias field and in-plane field magnitude) coincides with the center of the stable region indicated as PROPAGATE in figure 11(c). Bubble annihilation and generation may be accomplished by modifying the bias field at specific data sites. This is done by locating very tiny current loops in the data stream of the device as indicated in figures 11(a) and 11(b). For annihilation (erase), a current pulse is passed in a direction which increases the bias field beyond the stable operating region within the area of the current loop (Fig. 11(a)). For bubble generation, a current as shown in figure 11(b) produces a field which opposes the d.c. bias field, thus momentarily shifting the local bias field within the area of the current loop below the stable region. While the local field is depressed a bubble will spontaneously nucleate. The current loops are fabricated as photolithographically etched aluminum-copper patterns and are separated from the permalloy by a silicon dioxide spacing layer.



(a) CURRENT LOOP TO ANNIHILATE BUBBLE

(b) CURRENT LOOP TO GENERATE BUBBLE

(c) CONDITIONS FOR BUBBLE OPERATION

Figure 11.- Bubble device operation.

APPENDIX A

Bubbles are detected by being routed underneath a permalloy element which is connected to a bridge circuit suitable for monitoring the instantaneous resistance of the element. Since permalloy is magneto-resistive, its resistance will change as the magnetic field changes. Thus, the presence or absence of a bubble immediately underneath the detector element modifies the resistance of the element and these resistance changes serve to detect bubbles.

The bubble device architecture presently used by all manufacturers is the major track - minor loop scheme sketched in figure 12. The organization has an input track, an output track, and a large number of parallel minor storage loops. The two principal advantages of this architecture are fast access to blocks of data and redundant minor loops for increased device yield. Approximately 15% of the minor loops are redundant to allow for processing defects. Defects occur at random locations on the device and post-fabrication testing determines which loops are to be used. The code identifying (or mapping) the useable loops may be stored either on a separately accessible bootloop on the device or on an external memory device. The bootloop code is used when writing data to interject a "0" at locations in the data stream which correspond to the unused minor loops. Thus, user data is stored only in the predetermined good loops. Similarly, the bootloop code is used when reading data to ignore data positions corresponding to the unused minor loops.

Data is accessed in blocks equal to the number of useable minor loops. In writing, the data block is generated one bit at a time and the data propagated until all bits are positioned at the swap gates. A pulse of current in the swap gates simultaneously transfers the block of data into the

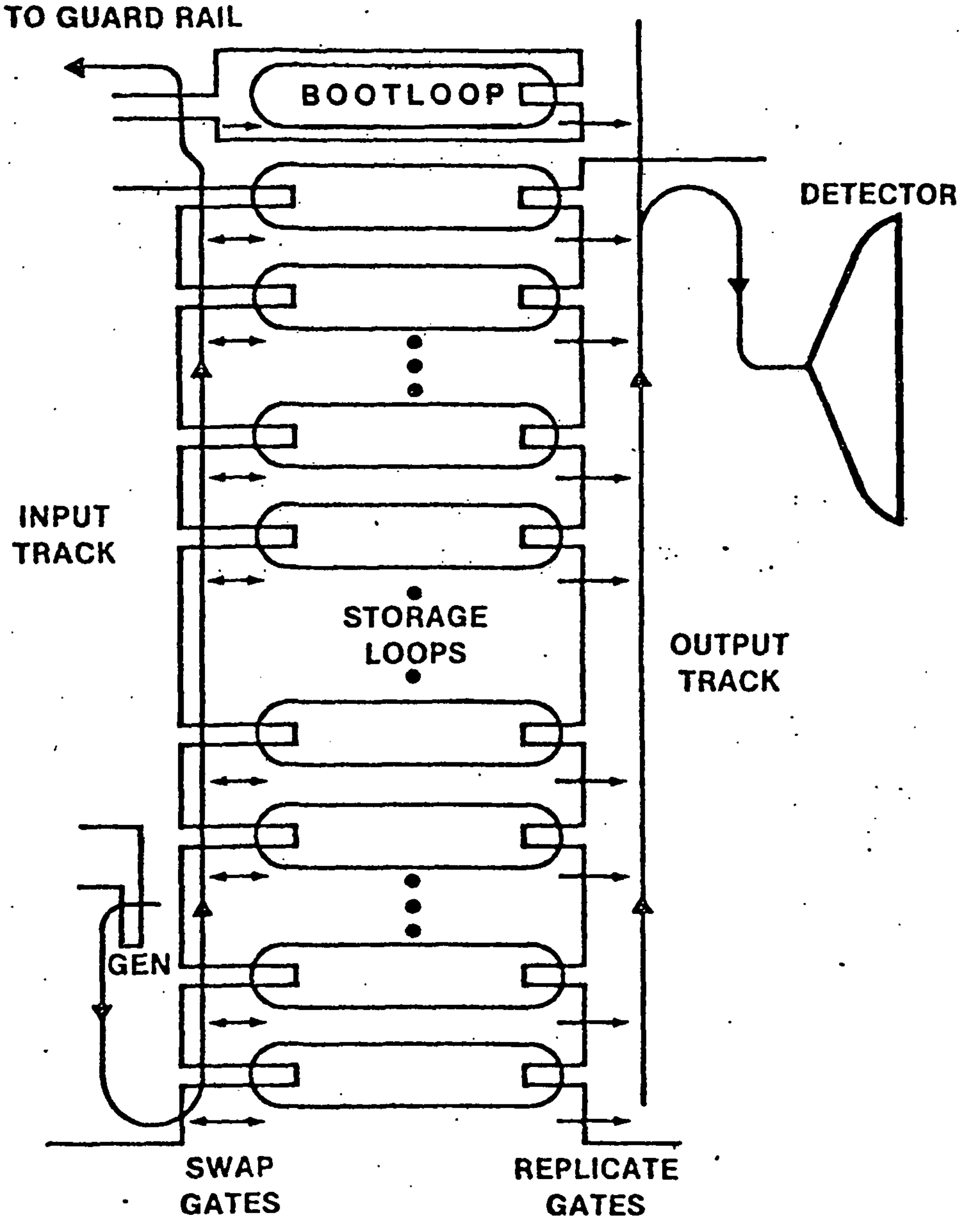


Figure 12 - Major track-minor loop device architecture.

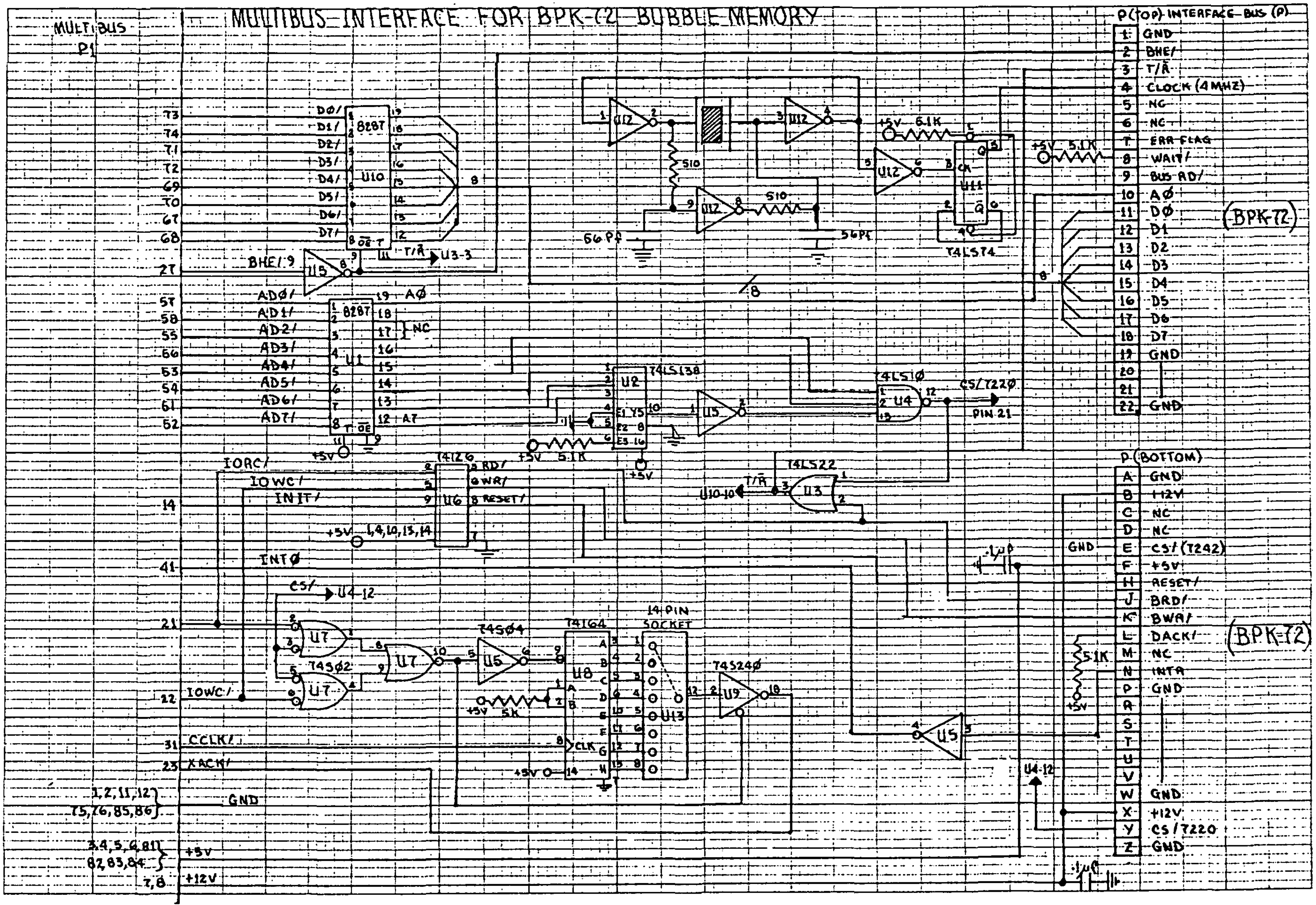
minor loops for storage and an equivalent data block out of the minor loops and onto the input track. The old data block is then propagated to a guard rail annihilator while the next block of data is being generated and brought into position for the next swap.

In detection, the desired data block is first positioned at the replicate gates. A pulse of current in the replicate gates causes the data block to be replicated onto the output track where it can be propagated into the detector. The data in the minor loop is not erased but is maintained indefinitely until new data is stored in its place as described in the write process above.

Although device densities are suitable now for onboard system development in the 10^8 bit capacity range and megabit data rates, there remains a significant potential for substantially increased capacity and data rate. Device data site period (and density) is limited by the minimum feature which can be photolithographically delineated. Ion implant technology (classification 1 of Fig. 8) offers an order of magnitude density increase over conventional permalloy for a given resolution and operational power. The self-structured current access technology (classification 4 of Fig. 8) offers an even further increase in system density by eliminating the in-plane field coils. Current access techniques also offer an order of magnitude increase in data rates. Ion implant and self-structured current access devices are under development for potential application in systems in the 10^9 bit and 10^{10} bit capacity ranges, respectively.

APPENDIX B

HARDWARE INTERFACE SCHEMATIC



MULTIBUS INTERFACE FOR BPK-72 BUBBLE MEMORY

NOTE: ALL EXTRA NOR GATES TIE TO +5V
" " INVERTERS TIE TO GND
" " NAND GATES TIE TO GND

AD

APPENDIX C

TEST SOFTWARE ROUTINES

APPENDIX C

The software routines contained in this appendix are the test driver routines necessary to utilize the Intel driver routines, get information to and from the terminal, and perform the fast initialization command. These routines, listed here in alphabetical order, are written in 8085 assembly language and PL/M-80. An outline of standard PL/M-80 program format is shown in figure 7.

```
M:DO              /*beginning of module*/
          external procedure declarations
          variable declarations
         PROCEDURE number_one;
                  DO
                  END;    /*number_one*/
         PROCEDURE number_two;
                  DO
                  END;    /*number_two*/
END M;   /*end of module*/
```

Figure 7. General PL/M-80 Format

The main controlling program, BMCOM, continually takes bubble memory commands from the user and allows them to be executed if they are valid commands. The module DOCMD contains the programs necessary to actually implement the commands. BMIO and IOMOD contain the programs which query the user about various transfer parameters and translate the requests for loading into the BMC. The program SETPAR sets up the initial values for the parametric registers. CKSTAT accesses the status of the controller after each operation and issues error messages when necessary. The fast initialization procedure is implemented in the FSTINT module. The other program modules listed here are for getting messages to and from the terminal. These programs are: MENU, CHKVAL, ERRMOD, TERMIO, and TSTMOD. A short description is provided at the beginning of each program. The Intel driver routines(see reference 4) that are referred to in these program listings are: ABORT, FIFORS, INBUBL, INTPAR, MBMPRG, RDBLRS, RDBOOT, RDBUBL, WRBLRM, WRBLRS, and WRBUBL.

```
$NOLIST
/****************************************************************************
DESCRIPTION:  EXTERNAL DECLARATIONS FOR THE MAIN PROGRAM MODULE
****************************************************************************/


     CI:PROCEDURE BYTE EXTERNAL;                              /*MONITOR*/
                END CI;
     PRINT$MENU:PROCEDURE EXTERNAL;                           /*MENU,PG. */
                END PRINT$MENU;
     INVALID$CMD:PROCEDURE EXTERNAL;                          /*ERRMOD,PG. */
                END INVALID$CMD;
     ECHO:PROCEDURE (CHAR) EXTERNAL;                          /*MONITOR*/
                DECLARE CHAR BYTE;
                END ECHO;
     CHK3:PROCEDURE EXTERNAL;                                 /*CKSTAT*/
                END CHK3;
     SEND$MESSAGE:PROCEDURE (PTR) EXTERNAL;                   /*TERMIO*/
                DECLARE PTR ADDRESS;
                END SEND$MESSAGE;
     EXECUTE$COMMAND:PROCEDURE (NUM) EXTERNAL;                /*DOCMD*/
                DELCARE NUM BYTE;
                END EXECUTE$COMMAND;
     CROUT:PROCEDURE EXTERNAL;                                /*MONITOR*/
                END CROUT;
     FETCH$CMD:PROCEDURE BYTE EXTERNAL;                       /*DOCMD*/
                END FETCH$CMD;
$LIST
```

```
$NOLIST
/****************************************************************************
DESCRIPTION:  VARIABLE DECLARATIONS
****************************************************************************/


     DECLARE (BLADDR,BLFSA) ADDRESS PUBLIC;
     DECLARE (BCREG,WR$RAM,RD$RAM) ADDRESS PUBLIC;
     DECLARE (ERR$FLAG,EXIT$FLAG,RW#FLAG) BYTE PUBLIC;
     DECLARE CMD$NUMBER BYTE PUBLIC;
     DECLARE FOREVER LITERALLY 'WHILE EXIT$FLAG=1';
$LIST
```

```
$NOLIST
/****************************************************************************
DESCRIPTION:  CONSOLE MESSAGES
****************************************************************************/


DECLARE CR LITERALLY '0DH',LF LITERALLY '0AH';
DECLARE MSG (*) BYTE DATA
    ('0 NORMAL INITIALIZATION  ',CR,
     '1 FAST INITIALIZATION    ',CR,
     '2 READ BUBBLE DATA       ',CR,
     '3 WRITE BUBBLE DATA      ',CR,
     '4 READ BOOTLOOP REG.     ',CR,
     '5 WRITE BOOTLOOP REG.    ',CR,
     '6 READ BOOTLOOP          ',CR,
     '7 RESET FIFO             ',CR,
     '8 MBM PURGE              ',CR,
     '9 ABORT                  ',CR,
     'A EXIT                   ',CR);
DECLARE MS0 (*) BYTE DATA
     ('ENTER THE COMMAND NUMBER',CR);
DECLARE MS1 (*) BYTE DATA
     ('INVALID COMMAND--TRY AGAIN',CR);
DECLARE MS2 (*) BYTE DATA
     ('OPERATION INCOMPLETE',CR);
DECLARE MS3 (*) BYTE DATA
     ('ENTER # OF PAGES TO BE TRANSFERRED--0 TO 255',CR);
DECLARE MS4 (*) BYTE DATA
     ('ENTER PAGE LOCATION IN BUBBLE MODULE--0 TO 2048',CR);
DECLARE MS5 (*) BYTE DATA
     ('ENTER A CARRIAGE RETUREN TO CONTINUE',CR);
DECLARE MS6 (*) BYTE DATA
     ('ABORT PHASE',CR);
DECLARE MS7 (*) BYTE DATA
     ('PURGE PHASE',CR);
DECLARE MS8 (*) BYTE DATA
     ('ILLEGAL ENTRY--TRY AGAIN',CR);
DECLARE MS9 (*) BYTE DATA
     ('CONTINUOUS READ OR WRITE?--Y/N',CR);
DECLARE MS10 (*) BYTE DATA
     ('INPUT AN "A" TO ABORT',CR);
DECLARE MS11 (*) BYTE DATA
     ('OPERATION ABORTED'CR);
DECLARE MS12 (*) BYTE DATA
     ('NOW EXECUTING COMMAND # ',CR);
DECLARE MS13 (*) BYTE DATA
     ('EXECUTING A CONTINUOUS READ',CR);
DECLARE MS14 (*) BYTE DATA
     ('EXECUTING A CONTINUOUS WRITE',CR);
DECLARE MS15 (*) BYTE DATA
     ('HAVE A NICE DAY',CR);
$LIST
```

```
ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE BMIO
OBJECT MODULE PLACED IN :F1:BMIO.OBJ
COMPILER INVOKED BY:  PLM80 :F1:BMIO.PLM  DEBUG



            $TITLE('BMIO--6/6/83')
 1          BMIO:DO;
            $INCLUDE(:F1:MSGMOD.LIT)
          = $NOLIST
20   1      ABORT:PROCEDURE EXTERNAL;          /*ABORT*/
21   2          END ABORT;
22   1      RDBUBL:PROCEDURE (BCREG,BLADDR) EXTERNAL;      /*BUBIOR*/
23   2          DECLARE (BCREG,BLADDR) ADDRESS;
24   2          END RDBUBL;
25   1      WRBUBL:PROCEDURE (BCREG,RAMAD) EXTERNAL;       /*BUBIOW*/
26   2          DECLARE (BCREG,RAMAD) ADDRESS;
27   2          END WRBUBL;
28   1      SETPAR:PROCEDURE (P,B) EXTERNAL;  /*PARMET*/
29   2          DECLARE (P,B) ADDRESS;
30   2          END SETPAR;
31   1      SEND$MESSAGE:PROCEDURE (PTR) EXTERNAL;         /*TERMIO*/
32   2          DECLARE PTR ADDRESS;
33   2          END SEND$MESSAGE;
34   1      QUICK$CI:PROCEDURE BYTE EXTERNAL; /*TERMIO*/
35   2          END QUICK$CI;
36   1      CHK3:  PROCEDURE EXTERNAL;         /*CKSTAT*/
37   2          END CHK3;
38   1      CHK2:  PROCEDURE EXTERNAL;         /*CKSTAT*/
39   2          END CHK2;
40   1      CHK1:  PROCEDURE EXTERNAL;
41   2          END CHK1;
42   1      DECLARE (BCREG,WR$RAM,RD$RAM) ADDRESS EXTERNAL; /*BMCOM*/
43   1      DECLARE (PAGE$NUM,BM$PAGE) ADDRESS EXTERNAL;    /*IOMOD*/
44   1      DECLARE RW$FLAG BYTE EXTERNAL;
45   1      DECLARE KEY BYTE;
46   1      DECLARE STOP LITERALLY '41H';
```

```
          $EJECT
 47   1   READ$BUBBLE:PROCEDURE PUBLIC;
          /*********************************************************************

          DESCRIPTION:  CONTINUOUSLY READS A GIVEN NUMBER OF PAGES
                        IN THE BUBBLE MODULE--UNTIL ABORTED BY THE USER

               INPUTS:  P--# OF PAGES TO READ
                        B==# FIRST PAGE TO BE READ IN THE BUBBLE MODULE

              OUTPUTS:  BUBBLE DATA TO RAM, APPROPRIATE MESSAGE TO CONSOLE

                CALLS:  RDBUBL,SEND$MESSAGE,SETPAR,ABORT,C1,CHK2

             DESTROYS:  B,C,D,E

          ********************************************************************/

 48   2        CALL SEND$MESSAGE(.MB13);
 49   2        KEY-00H;
 50   2        DO WHILE KEY<>STOP;           /*READ TILL USER ABORT*/
 51   3             CALL SETPAR(PAGE$NUM:BM$PAGE); /*SET UP PARA REG DATA*/
 52   3             CALL RDBUBL(BCREG,RD$RAM); /*READ BUBBLE DATA*/
 53   3             CALL CHK2;                /*CHECK STATUS*/
 54   3             CALL ABORT;               /*END READ CMD*/
 55   3             KEY=QUICK$CI;                /*CHECK FOR USER ABORT*/
 56   3        END;
 57   2        CALL ABORT;
 58   2        CALL SEND$MESSAGE(.MS11);     /*SEND ABORT MSG TO CRT*/
 59   2        RW$FLAG=0;
 60   2   END READ$BUBBLE;
```

```
        $EJECT
61  1   WRITE$BUBBLE:PROCEDURE PUBLIC;

        /*******************************************************************

        DESCRIPTION:  CONTINUOUSLY WRITES A GIVEN NUMBER OF PAGES
                      IN THE BUBBLE MODULE--UNTIL ABORTED BY THE USER

             INPUTS:  P--# OF PAGES TO WRITTEN
                      B==# FIRST PAGE IN THE BUBBLE MODULE TO BE WRITTEN TO

            OUTPUTS:  DATA TO THE BUBBLE MODULE, APPROPRIATE MESSAGE TO THE
                      CONSOLE

              CALLS:  WRBUBL,SEND$MESSAGE,CI,SETPAR,ABORT,CHK1

           DESTROYS:  B,C,D,E

        *******************************************************************/

62  2       CALL SEND$MESSAGE(.M214);
63  2       KEY-00H;
64  2       DO WHILE KEY<>STOP;          /*WRITE TILL USER ABORT*/
65  3           CALL SETPAR(PAGE$NUM:BM$PAGE); /*SET UP PARA REG DATA*/
66  3           CALL WRBUBL(BCREG,WR$RAM); /*WRITE BUBBLE DATA*/
67  3           CALL CHK1;               /*CHECK STATUS*/
                CALL ABORT;              /*ABORT WRITE CMD*/
68  3           KEY=QUICK$CI;               /*CHECK FOR USER ABORT*/
69  3       END;
70  2       CALL ABORT;
71  2       CALL SEND$MESSAGE(.MS11);    /*SEND ABORT MSG TO CRT*/
72  2       RW$FLAG=0;
73  2   END WRITE$BUBBLE;
74  1   END BMIO;


MODULE INFORMATION:

    CODE AREA SIZE     = 0351H   849D
    VARIABLE AREA SIZE = 0001H     1D
    MAXIMUM STACK SIZE = 0002H     2D
    147 LINES READ
    0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE CHKVAL
OBJECT MODULE PLACED IN :F1:CHKVAL.OBJ
COMPILER INVOKED BY:  PLM80 :F1:CHKVAL.PLM DEBUG


            $TITLE('CHKVAL')
            $DATE(1/7/83)
1           CHKVAL:DO;
2    1          ILLEGAL$ENTRY:PROCEDURE EXTERNAL;          /*ERRMOD*/
3    2              END ILLEGAL$ENTRY;
4    1          DECLARE ENTRY$FLAG BYTE EXTERNAL;          /*IOMOD*/
5    1          DECLARE INDEX BYTE EXTERNAL;               /*TERMIO*/
6    1          DECLARE VALID$CHAR(256) BYTE;
7    1          DECLARE BUFFER(256) BYTE EXTERNAL;         /*TERMIO*/
8    1          DECLARE N BYTE;
```

```
        $EJECT
 9  1   PAGE$CHECK:PROCEDURE;

        /********************************************************************

        DESCRIPTION:  VERIFIES THE NUMBER OF PAGES TO BE TRANSFERRED

             INPUTS:  CHARACTERS IN THE INPUT BUFFER

            OUTPUTS:  ERROR MESSAGE TO CONSOLE FOR AN INVALID ENTRY

              CALLS:  ILLEGAL$ENTRY

           DESTROYS:  NONE

        ********************************************************************/

10  2        IF INDEX>3 THEN CALL ILLEGAL$ENTRY;
             ELSE
12  2           DO CASE INDEX-1;
13  3              IF VALID$CHAR(0) THEN CALL ILLEGAL$ENTRY;
                   IF VALID$CHAR(0) OR VALID$CHAR(1) THEN CALL ILLEGAL$ENTRY;
                   DO;
18  4                 IF BUFFER(0)<30H OR BUFFER(0)>32H THEN
19  4                    CALL ILLEGAL$ENTRY;
                      ELSE
20  4                    DO;
21  5                       IF BUFFER(0)=32H THEN
22  5                       DO;
23  6                            IF BUFFER(1)<30H OR BUFFER(1)>35H THEN
24  6                               CALL ILLEGAL$ENTRY
                             ELSE
25  6                               IF BUFFER(1)=35H AND (BUFFER(2)<30H OR
                                    BUFFER(2)>35H)
26  6                                  CALL ILLEGAL$ENTRY;
27  6                         END;
                            ELSE
28  5                         IF VALID$CHAR(1) OR VALID$CHAR(2) THEN
29  5                               CALL ILLEGAL$ENTRY;
30  5                    END;
31  4                 END;
32  3              END;
33  2   END PAGE$CHECK;
```

```
        $EJECT
34  1   CHECK$BM$PAGE:PROCEDURE;

        /*******************************************************************

        DESCRIPTION:  VERIFIES THE PAGE NUMBER REQUESTED IN THE BUBBLE MODULE

             INPUTS:  CHARACTERS IN THE INPUT BUFFER

            OUTPUTS:  ERROR MESSAGE TO CONSOLE FOR AN INVALID ENTRY

              CALLS:  ILLEGAL$ENTRY

           DESTROYS:  NONE

        *******************************************************************/

35  2       IF INDEX>4 THEN CALL ILLEGAL$ENTRY;
            ELSE
37  2          DO CASE INDEX-1;
38  3             IF VALID$CHAR(0) THEN CALL ILLEGAL$ENTRY;
                  IF VALID$CHAR(0) OR VALID$CHAR(1) THEN
41  3             CALL ILLEGAL$ENTRY;
                  IF VALID$CHAR(0) OR VALID$CHAR(1) OR VALID$CHAR(2) THEN
43  3             CALL ILLEGAL$ENTRY;
                  DO;
45  4             IF BUFFER(0)<30H OR BUFFER(0)>32H THEN
46  4                CALL ILLEGAL ENTRY;
                  ELSE
47  4                DO;
48  5                IF BUFFER(0)=32H THEN
49  5                   DO;
50  6                   IF BUFFER(1)<>30H THEN CALL ILLEGAL$ENTRY;
                        ELSE
52  6                      IF BUFFER(2)<30H OR BUFFER(2)>34H THEN
53  6                      CALL ILLEGAL$ENTRY;
                           ELSE
54  6                      IF BUFFER(2)=34H AND BUFFER(3)<30H
                                 OR BUFFER(3)>38H THEN
                                 CALL ILLEGAL$ENTRY;
55  6                     END;
56  6                END;
57  5             END;
58  4          END;
59  3       END;
60  2   END CHECK$BM$PAGE;
```

```
        $EJECT
61  1   CHECK$ENTRY$VALUE:PROCEDURE PUBLIC;

        /*******************************************************************

        DESCRIPTION:  VERIFIES THE INFORMATION THAT THE USER ENTERED AT THE
                      CONSOLE

             INPUTS:  INPUT BUFFER

            OUTPUTS:  ERROR MESSAGE TO CONSOLE FOR AN INVALID ENTRY

              CALLS:  PAGE$CHECK,CHECK$BM$PAGE

           DESTROYS:  NONE

        *******************************************************************/

62  2        DO N=0 TO (INDEX-1);
63  3           VALID$CHAR(N)=BUFFER(N)<30H OR BUFFER(N)>39H;
64  3        END;
65  2        DO CASE ENTRY$FLAG;
66  3           CALL PAGE$CHECK
67  3           CALL CHECK$BM$PAGE;
68  3        END;
69  2   END CHECK$ENTRY$VALUE;
70  1   END CHKVAL;


MODULE INFORMATION:

    CODE AREA SIZE     = 0131H   481D
    VARIABLE AREA SIZE = 0101H   257D
    MAXIMUM STACK SIZE = 0006H     6D
    121 LINES READ
    0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
```

```
ASM80 :FI:CKSTAT.S MACROFILE PAGEWIDTH(100) DEBUG MOD85

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1     CKSTAT   PAGE 1
CKSTAT--3/10/83

LOC OBJ                LINE           SOURCE STATEMENT

                          1  $TITLE('CKSTAT--3/10/83')
                          2  NAME CKSTAT
                          3  EXTRN ERROR
                          4  ;
                          5  ;**********************************************************
                          6  ;
                          7  ;DESCRIPTION:  CHECKS THE STATUS REGISTER OF THE 7220
                          8  ;
                          9  ;      INPUTS:  NONE
                         10  ;
                         11  ;     OUTPUTS:  ERROR MESSAGE WHEN APPROPRIATE
                         12  ;
                         13  ;       CALLS:  ERROR
                         14  ;
                         15  ;    DESTROYS:  NONE
                         16  ;
                         17  ;*********************************************************/
                         18                 CSEG
                         19  ;
                         20                 PUBLIC CHK1    ;CHECK WRITE OPERATION
0000 C5                  21 CHK1:           PUSH B         ;SAVE B REG
0001 47                  22                 MOV B,A        ;SAVE A REG
0002 EE40                23                 XRI 40H        ;SUCCESSFUL OPERATION??
0004 CA0D00   C          24                 JZ OKAY1       ;IF YES, RETURN
0007 78                  25                 MOV A,B        ;RESTORE A REG
0008 EE42                26                 XRI 42H        ;IF NOT, CHECK OTHER VALUE
000A C40000   E          27                 CNZ ERROR      ;SEND MESSAGE FOR ERROR
000D C1                  28 OKAY1:          POP B          ;RESTORE B REG
000E C9                  29                 RET            ;RETURN
                         30 ;
                         31                 PUBLIC CHK2    ;CHECK READ OPERATION
000F C5                  32 CHK2:           PUSH B         ;SAVE B REG
0010 47                  33                 MOV B,A        ;SAVE A REG
0011 EE40                34                 XRI 40H        ;SAME AS CHK1
0013 CA1C00   C          35                 JZ OKAY2
0016 78                  36                 MOV A,B
0017 EE48                37                 XRI 48H
0019 C40000   E          38                 CNZ ERROR
001C C1                  39 OKAY2:          POP B
001D C9                  40                 RET
                         41 ;
                         42                 PUBLIC CHK3    ;CHECK OTHER OPERATIONS
001E EE40                43 CHK3:           XRI 40H        ;SAME AS CHK1 AND CHK2
0020 C40000   E          44                 CNZ ERROR
0023 C9                  45                 RET
                         46 ;
                         47                 END
PUBLIC SYMBOLS
CHK 1   C 0000   CHK2   C 000F   CHK3   C 001E
```

```
ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE CNVERT
OBJECT MODULE PLACED IN :F1:CNVERT.OBJ
COMPILER INVOKED BY:  PLM80 :F1:CNVERT.PLM DEBUG


         $TITLE('CNVERT--4/29/83')
 1       CNVERT:DO;
 2   1       DECLARE BUFFER(256) BYTE EXTERNAL;     /*TERMIO*/
 3   1       DECLARE DUMBUF BYTE;
 4   1       DECLARE N BYTE;
 5   1       DECLARE X ADDRESS;
 6   1       DECLARE OR LITERALLY 'ODH';
 7   1   ASCOO$TO$BINARY: PROCEDURE ADDRESS PUBLIC;

         /********************************************************************

         DESCRIPTION:  CONVERTS CHARACTER FROM ASCII TO BINARY

              INPUTS:  ASCII CHARACTERS IN A BUFFER

             OUTPUTS:  BINARY EQUIVALENT

               CALLS:  NONE

            DESTROYS:  NONE

         ********************************************************************/

 8   2       X=0;
 9   2       N=0;
10   2       DO WHILE BUFFER(N) <> CR;
11   3           DUMBUF=BUFFER(N)-30H;
12   3           X=10*X + DUMBUF;
13   3           N=N+1;
14   3       END;
15   2       RETURN X;
16   2   END ASCII$TO$BINARY;
17   1   END CNVERT;


MODULE INFORMATION:

     CODE AREA SIZE     = 0040H   64D
     VARIABLE AREA SIZE = 0004H    4D
     MAXIMUM STACK SIZE = 0004H    4D
     31 LINES READ
     0 PROGRAM ERRORS


END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE DOCMD
OBJECT MODULE PLACED IN :F1:DOCMD.OBJ
COMPILER INVOKED BY:  PLM80 :F1:DOCMD.PLM DEBUG


          $TITLE('DOCMD--6/6/83')
 1        DOCMD:DO;
          $INCLUDE(:F1:PRMOD2.EXT)
        = $NOLIST
53   1         DECLARE (PAGE$NUM,BM$PAGE) ADDRESS EXTERNAL;   /*IOMOD*/
54   1         DECLARE (BLADDR,BLFSA) ADDRESS EXTERNAL;       /*BMCOM*/
55   1         DECLARE (BCREG,WR$RAM,RD$RAM) ADDRESS EXTERNAL;  /*BMCOM*/
56   1         DECLARE (EXIT$FLAG,RW$FLAG) BYTE EXTERNAL;     /*BMCOM*/
```

```
$NOLIST
/*******************************************************************************
DESCRIPTION:  EXTERNAL DECLARATIONS FOR THE DOCND MODULE
*******************************************************************************/
     ECHO:PROCEDURE (CHAR) EXTERNAL;                          /*MONITOR*/
          DECLARE CHAR BYTE;
          END ECHO;
     CROUT:PROCEDURE EXTERNAL;                                /*MONITOR*/
          END CROUT;
     BMINIT:PROCEDURE EXTERNAL;                               /*INIT*/
          END BMINIT;
     INBUBL:PROCEDURE EXTERNAL;                               /*INIT*/
          END INBUBL;
     ABORT:PROCEDURE EXTERNAL;                                /*ABORT*/
          END ABORT;
     FAST$INITIALIZATION:PROCEDURE EXTERNAL;                  /*FSTINT*/
          END FAST$INITIALIZATION;
     FIFORS:PROCEDURE EXTERNAL;                               /*FIFO*/
          END FIFORS;
     RDBUBL:PROCEDURE (BCREG,RAMAD) EXTERNAL;                 /*BUBIOR*/
          DECLARE (BCREG,RAMAD) ADDRESS;
          END RDBUBL;
     WRBUBL:PROCEDURE (BCREG,ROMAD) EXTERNAL;                 /*BUBIOW*/
          DECLARE (BCREG,ROMAD) ADDRESS;
          END WRBUBL;
     READ$BUBBLE:PROCEDURE EXTERNAL;                          /*RDWR*/
          END READ$BUBBLE;
     WRITE$BUBBLE:PROCEDURE EXTERNAL;                         /*RDWR*/
          END WRITE$BUBBLE;
     RDBLRS:PROCEDURE (BCREG,BLADDR) EXTERNAL;                /*BTLOOP*/
          DECLARE (BCREG,BLADDR) ADDRESS;
          END RDBLRS;
     WRBLRS:PROCEDURE (BCREG,RAMAD) EXTERNAL;                 /*BTLOOP*/
          DECLARE (BCREG,RAMAD) ADDRESS;
          END WRBLRS;
     MBMPRG:PROCEDURE EXTERNAL;                               /*BUBIOW*/
          END MBMPRG;
     RDBOOT:PROCEDURE (BCREG,BLADDR) EXTERNAL;                /*BTLOOP*/
          DECLARE (BCREG,BLADDR) ADDRESS;
          END RDBOOT;
     IO$DEFINITION:PROCEDURE EXTERNAL;                        /*IOMOD*/
          END 10$DEFINITION;
     CHK1:PROCEDURE EXTERNAL;                                 /*CKSTAT*/
          END CHK1;
     CHK2:PROCEDURE EXTERNAL;                                 /*CKSTAT*/
          END CHK2;
     CHK3:PROCEDURE EXTERNAL;                                 /*CKSTAT*/
          END CHK3;
     SETPAR:PROCEDURE (BC,DE) EXTERNAL;                       /*PARMET*/
          DECLARE (BC,DE) ADDRESS;
          END SETPAR;
     INTPAR:PROCEDURE EXTERNAL;                               /*PARMET*/
          END INTPAR;
     CI:PROCEDURE BYTE EXTERNAL;                              /*MONITOR*/
          END CI;
$LIST
```

```
PL/M-80 COMPILER   BMCOM--9/16/83                              PAGE 1

ISIS-11 PL/M-80 V4.0 COMPILATION OF MODULE BMCOM
OBJECT MODULE PLACED IN :F1:BMCOM.OBJ
COMPILER INVOKED BY:  PLM80 :F1:BMCOM.PLM DEBUG

              $TITLE('BMCOM--9/16/83')
1             BMCOM:DO;
              /*****************************************************************
              DESCRIPTION:  MAIN PROGRAM TO INITIALIZE THE SBC 80/24 AND THE
                            BMC 7220.

                   INPUTS:  USER COMMANDS FROM A CONSOLE

                  OUTPUTS:  COMMANDS TO THE BMC 7220 AND MESSAGES TO THE CONSOLE

                    CALLS:  PRINT$MENU,ECHO,EXECUTE$COMMAND,INVALID$COMMAND

                 DESTROYS:  NONE

              *****************************************************************/
              $INCLUDE(:F1:PRMOD1.EXT)
            = $NOLIST
              $INCLUDE(:F1:DECMOD.LIT)
            = $NOLIST
              $INCLUDE(:F1:MSGMOD.LIT)
            = $NOLIST
46    1           MAIN:PROCEDURE PUBLIC;
47    2               BCREG=7300H;          /*LOCATION OF PARA. REG. DATA*/
48    2               BLADDR=7800H;         /*LOCATION TO WRITE BTLOOP DATA TO*/
49    2               BLFSA=7800H;          /*BTLOOP DATA IN ROM-(TEMP IN RAM)-*/
50    2               WR$RAM=7255H          /*START ADDR. OF DATA TO BE WRITTEN*/
51    2               RD$RAM=792BH;         /*  "     "    "    "   " " READ*/
52    2               EXIT$FLAG=1;
53    2               DO FOREVER;      /*LOOP UNTIL EXIT COMMAND RECEIVED*/
54    3                   ERR$FLAG=1;
55    3                   RW$FLAG=0;L
56    3                   CALL PRINT$MENU;
57    3                   CMD$NUMBER=FETCH$CMD;
58    3                   IF CMD$NUMBER<0 OR CMD$NUMBER >10 THEN
59    3                   CALL INVALID$CMD;
                          ELSE
60    3                   DO;
61    4                       CALL EXECUTE$COMMAND(CMD$NUMBER);
62    4                   END;
63    3               END;
64    2               CALL SEND$MESSAGE(.MS15);
65    2           END MAIN;
66    1       END BMCOM;


MODULE INFORMATION:

    CODE AREA SIZE     = 0329H   809D
```

```
          $EJECT
57   1    FETCH$CMD: PROCEDURE BYTE PUBLIC;
         /*******************************************************************

          DESCRIPTION:  GETS THE COMMAND NUMBER FROM THE TABLE VALUE
                        ENTERED BY THE USER

               INPUTS:  COMMAND FROM USER

              OUTPUTS:  APPROPRIATE COMMAND NUMBER FOR USE BY THE
                        EXECUTE$COMMAND ROUTINE

                CALLS:  CI,ECHO,CROUT

             DESTROYS:  A

          *******************************************************************/

58   2          DECLARE CMD BYTE;
59   2          CMD=(CI AND 7FH) - 30H;
60   2          CALL ECHO(CMD + 30H);
61   2          CALL CROUT;
62   2          IF CMD >9 THEN CMD=CMD - 7;
64   2          RETURN CMD;
65   2    END FETCH$CMD;
```

```
         $EJECT
66   1   EXECUTE$COMMAND: PROCEDURE (NUM) PUBLIC;
        /*****************************************************************
        DESCRIPTION:  CALLS THE APPROPRIATE ROUTINE TO EXECUTE THE COMMAND
                      GIVEN BY THE USER

             INPUTS:  CMD$NUMBER

            OUTPUTS:  NONE

              CALLS:  BMINIT,INBUBL,FSTINT,IO$DEFINITION,RDBUBL,CHK2,WRBUBL,
                      CHK1,RDBLRS,WRBLRS,ABORT,RDBOOT,FIFORS,MBMPRG,
                      READ$BUBBLE,WRITE$BUBBLE,SETPAR

           DESTROYS:  B,C,D,E,H,L,A
        *****************************************************************/

67   2       DECLARE NUM BYTE;
68   2       DO CASE NUM;
69   3           DO;
70   4               CALL BMINIT;                        /*NORMAL INIT*/
71   4               CALL INBUBL;
72   4               CALL CHK3;
73   4           END;
74   3           CALL FAST$INITIALIZATION;           /*FSTINT*/
75   3           DO;                     /*READ BUBBLE DATA*/
76   4                   CALL IO$DEFINITION;
77   4               IF RW$FLAG=1 THEN
78   4               CALL READ$BUBBLE;                   /*CONTINUOUS READ*/
                     ELSE
79   4               DO;
80   5                   CALL SETPAR(PAGE$NUM,BM$PAGE);
81   5                       CALL RDBUBL(BCREG,RD$RAM);
82   5                       CALL CHK2;
83   5               END;
84   4           END;
85   3           DO;                     /*WRITE BUBBLE DATA*/
86   4               CALL IO$DEFINITION;
87   4               IF RW$FLAG=1 THEN
88   4                CALL WRITE$BUBBLE;                 /*CONTINUOUS WRITE*/
                     ELSE
89   4               DO;
90   5                   CALL SETPAR(PAGE$NUM,BM$PAGE);
91   5                       CALL WRBUBL(BCREG,WR$RAM);
92   5                       CALL CHK1;
93   5               END;
94   4           END;
95   3           DO;
96   4               CALL RDBLRS(BCREG,BLADDR);      /*READ BOOTLOOP REG*/
97   4               CALL CHK3;      /*CHECK STATUS*/
98   4           END;
```

```
          $EJECT
 99  3         DO;
100  4              CALL WRBLRS(BCREG,BLFSA);   /*WRITE BOOTLOOP REG*/
101  4              CALL CHK3;        /*CHECK STATUS*/
102  4         END;
103  3         DO;
104  4              CALL RDBOOT(BCREG,BLADDR);  /*READ BOOTLOOP*/
105  4              CALL CHK3;        /*CHECK STATUS*/
106  4         END;
107  3         DO;
108  4              CALL FIFORS;                /*RESET FIFO*/
109  4              CALL CHK3;        /*CHECK STATUS*/
110  4         END;
111  3         DO;
112  4              CALL MBMPRG;                /*PURGE*/
113  4              CALL CHK3;        /*CHECK STATUS*/
114  4         END;
115  3         DO;
116  4              CALL ABORT;       /*ABORT*/
117  4              CALL CHK3;        /*CHECK STATUS*/
118  4         END;
119  3         EXIT$FLAG=0;
120  3      END;
121  2    END EXECUTE$COMMAND;
122  1    END DOCMD;



MODULE INFORMATION:

     CODE AREA SIZE     = 011FH   287D
     VARIABLE AREA SIZE = 0002H     2D
     MAXIMUM STACK SIZE = 0002H     2D
     167 LINES READ
     0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER   ERRMOD--6/2/83                                PAGE 1

ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE ERRMOD
OBJECT MODULE PLACED IN :F1:ERRMOD.OBJ
COMPILER INVOKED BY:  PLM80 :F1:ERRMOD.PLM DEBUG

              $TITLE('ERRMOD--6/2/83')
1             ERRMOD:DO;
             /*****************************************************************
             DESCRIPTION:  OUTPUTS ERROR MESSAGES TO THE TERMINAL

                  INPUTS:  NONE

                 OUTPUTS:  THE APPROPRIATE MESSAGE

                   CALLS:  SEND$MESSAGE,CI

                DESTROYS:  NONE
             *****************************************************************/
 2   1            CI: PROCEDURE BYTE EXTERNAL           /*MONITOR*/
 3   2                END CI;
 4   1            SEND$MESSAGE: PROCEDURE (PTR) EXTERNAL;   /*TERMIO*/
 5   2                DECLARE PTR ADDRESS;
 6   2                END SEND$MESSAGE;
              INCLUDE(:F1:MSGMOD.LIT)
            = $NOLIST
25   1            DECLARE (ERR$FLAG,RW$FLAG) BYTE EXTERNAL;  /*BMCOM*/
26   1            WAIT: PROCEDURE;
27   2                CALL SEND$MESSAGE(.MS5);
28   2                DO WHILE (CI AND 7FH) <> CR;    /*WAIT FOR USER TO*/
29   3                END;                  /*ENTER CARRIAGE RETURN*/
30   2                ERR$FLAG=1;
31   2            END WAIT;
32   1        INVALID$CMD: PROCEDURE PUBLIC;          /***INVALID COMMAND MSG**/
33   2                CALL SEND$MESSAGE(.MS1);
34   2                CALL WAIT;
35   2            END INVALID$CMD;
36   1        ERROR: PROCEDURE PUBLIC;             /**OPERATION INCOMPLETE MSG**/
37   2                CALL SEND$MESSAGE(.MS2);
38   2                IF RW$FLAG=1 THEN CALL WAIT;
40   2            END ERROR;
41   1        ILLEGAL$ENTRY: PROCEDURE PUBLIC;   /*MSG--USER ENTERED WRONG INFO*/
42   2                CALL SEND$MESSAGE(.MS8);
43   2                CALL WAIT;
44   2            END ILLEGAL$ENTRY;
45   1        END ERRMOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 0301H   769D
     VARIABLE AREA SIZE = 0000H     0D
     MAXIMUM STACK SIZE = 0004H     4D
```

```
ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE FSTINT
OBJECT MODULE PLACED IN :F1:FSTINT.OBJ
COMPILER INVOKED BY: PLM80 :F1:FSTINT.PLM DEBUG


             $TITLE('FSTINT--8/26/83')
 1           FSTINT: DO;
             $INCLUDE(:F1:MSGMOD.LIT)
           = $NO LIST
20   1            ABORT: PROCEDURE EXTERNAL;        /*ABORT*/
21   2                 END ABORT;
22   1            CHK3: PROCEDURE EXTERNAL;         /*CKSTAT*/
23   2                 END CHK3;
24   1            MBMPRG: PROCEDURE EXTERNAL;       /*BUBIOW*/
25   2                 END MBMPRG;
26   1            SEND$MESSAGE: PROCEDURE (PRT) EXTERNAL;  /*TERMIO*/
27   2                 DECLARE PRT ADDRESS;
28   2                 END SEND$MESSAGE;
29   1            WRBLRM: PROCEDURE (BCREG,BLADDR) EXTERNAL;  /*BTLOOP*/
30   2                 DECLARE (BCREG,BLADDR) ADDRESS;
31   2                 END WRBLRM;
32   1            TEST:PROCEDURE EXTERNAL;          /*TSTMOD*/
33   2                 END TEST;
34   1            BMINIT:PROCEDURE EXTERNAL;        /*INIT*/
35   2                 END BMINIT;
36   1            INTPAR: PROCEDURE EXTERNAL;       /*PARMET*/
37   2                 END INTPAR;
38   1            DECLARE ERR$FLAG BYTE EXTERNAL;   /*BMCOM*/
39   1            DECLARE (BCREG,BLFSA) ADDRESS EXTERNAL;   /*BMCOM*/
```

```
          $EJECT
40    1   FAST$INITIALIZATION: PROCEDURE PUBLIC;
         /*******************************************************************
         DESCRIPTION:  INITIALIZES THE BUBBLE MODULE BY FETCHING THE
                       BOOT LOOP INFORMATION FROM AN EXTERNAL EPROM

              INPUTS:  NONE

             OUTPUTS:  PROPER PARAMETERS AND THE BOOT LOOP DATA TO THE
                       7220 BMC

               CALLS:  ABORT,MBMPRG,CHK3,BMINIT,INTPAR,TEST

            DESTROYS:  B,C,D,E
         *******************************************************************/

41    2             CALL TEST;
42    2             CALL ABORT;                  /*SEND ABORT TO BM*/
43    2             CALL CHK3;                   /*CHECK STATUS*/
44    2             IF ERR$FLAG <> 1 THEN        /*PROCEED IF SUCCESSFUL*/
45    2             DO;
46    3                 CALL MBMPRG;             /*PURGE PARA. REG. DATA*/
47    3                 CALL CHK3;               /*CHECK STATUS*/
48    3                 IF ERR$FLAG <> 1 THEN    /*PROCEED IF SUCCESSFUL*/
49    3                    DO;
50    4                 CALL BMINIT;             /*SET PARA REG DATA*/
51    4                 CALL INTPAR;             /*LOAD PARA REGS*/
52    4                        CALL WRBLRM(BCREG,BLFSA); /*LOAD BL REG.*/
53    4                 CALL CHK3;               /*CHECK STATUS*/
54    4                   END;
                            ELSE
55    3                       CALL SEND$MESSAGE(.MS7); /*MSG--PURGE INCMPLT*/
56    3             END;
                    ELSE
57    2                 CALL SEND$MESSAGE(.MS6);  /*MSG--ABORT INCMPLT*/
58    2        END FAST$INITIALIZATION;
59    1   END FSTINT;


MODULE INFORMATION:

     CODE AREA SIZE     = 0309H     777D
     VARIABLE AREA SIZE = 0000H       0D
     MAXIMUM STACK SIZE = 0002H       2D
     115 LINES READ
     0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE IOMOD
OBJECT MODULE PLACED IN :F1:IOMOD.OBJ
COMPILER INVOKED BY: PLM80 :F1:IOMOD.PLM DEBUG


             $TITLE('IOMOD--5/25/83')
1            IOMOD: DO;
             $INCLUDE(:F1:MSGMOD.LIT)
           = $NOLIST
20   1            SEND$MESSAGE: PROCEDURE (PTR) EXTERNAL;   /*TERMIO*/
21   2                 DECLARE PTR ADDRESS;
22   2                 END SEND$MESSAGE;
23   1            SETPAR: PROCEDURE (PAGE$NUM,BM$PAGE) EXTERNAL; /*PARMET*/
24   2                 DECLARE (PAGE$NUM,BM$PAGE) ADDRESS;
25   2                 END SETPAR;
26   1            READ$CONSOLE: PROCEDURE EXTERNAL;         /*TERMIO*/
27   2                 END READ$CONSOLE;
28   1            ASCII$TO$BINARY: PROCEDURE ADDRESS EXTERNAL; /*CNVERT*/
29   2                 END ASCII$TO$BINARY;
30   1            ECHO:PROCEDURE (CHAR) EXTERNAL;           /*MONITOR*/
31   2                 DECLARE CHAR BYTE;
32   2                 END ECHO;
33   1            ILLEGAL$ENTRY:PROCEDURE EXTERNAL;         /*ERRMOD*/
34   2                 END ILLEGAL$ENTRY
35   1            CI:PROCEDURE BYTE EXTERNAL;               /*MONITOR*/
36   2                 END CI;
37   1            CROUT:PROCEDURE EXTERNAL;                 /*MONITOR*/
38   2                 END CROUT;
39   1            CHECK$ENTRY$VALUE:PROCEDURE EXTERNAL;        /*CHKVAL*/
40   2                 END CHECK$ENTRY$VALUE;
41   1            DECLARE (PAGE$NUM,BM$PAGE) ADDRESS PUBLIC;
42   1            DECLARE BUFFER(256) BYTE EXTERNAL;        /*TERMIO*/
43   1            DECLARE (ERR$FLAG,RW$FLAG) BYTE EXTERNAL;    /*BMCOM*/
44   1            DECLARE YES LITERALLY '59H',NO LITERALLY '43H';
45   1            DECLARE ENTRY$FLAG BYTE PUBLIC;
46   1            DECLARE TWO$CHANNELS LITERALLY '1000H';
```

```
          $EJECT
47   1    FETCH$NUMBER: PROCEDURE ADDRESS;
         /*******************************************************************

         DESCRIPTION:  FETECHES INPUT FROM THE TERMINAL AND CONVERTS TO
                       THE DECIMAL EQUIVALENT

              INPUTS:  DATA FROM THE TERMINAL

             OUTPUTS:  NONE

               CALLS:  READ$CONSOLE,ASCII$TO$DECIMAL

            DESTROYS:  NONE

         *******************************************************************/

48   2         DECLARE TEMP ADDRESS
49   2         ERR$FLAG=0;
50   2         TEMP=000H;
51   2         CALL READ$CONSOLE;
52   2         CALL CHECK$ENTRY$VALUE;
53   2         IF ERR$FLAG=0 THEN TEMP=ASCII$TO$BINARY;
55   2         RETURN TEMP;
56   2    END FETCH$NUMBER;
```

```
PL/M-80 COMPILER   IOMOD--5/25/83                               PAGE 3

            $EJECT
 57   1     IO$DEFINITION: PROCEDURE PUBLIC;
           /*******************************************************************
           DESCRIPTION:  ALLOWS THE USER TO DEFINE THE I/O INFORMATION
                         FOR THE BUBBLE MODULE FROM THE TERMINAL

                INPUTS:  PAGE$NUM--NUMBER OF PAGES TO BE TRANSFERRED
                         BM$PAGE--LOCATION IN THE BUBBLE MODULE

               OUTPUTS:  PAGE$NUM AND BM$PAGE TO THE PROPER LOCATION
                         IN THE RAM FOR LOADING IN THE 7220 BMC
                         PARAMETRIC REGISTERS,ERROR MESSAGES TO USER

                 CALLS:  SEND$MESSAGE,READ$CONSOLE,SETPAR,FETCH$NUMBER,CI
                         ILLEGAL$ENTRY

              DESTROYS:  B,C,D,E,H,L
           *******************************************************************/

 58   2          DECLARE KEY BYTE;
 59   2               AGAIN1:  CALL SEND$MESSAGE(.MS3);
 60   2                ENTRY$FLAG=0;
 61   2                    PAGE$NUM=FETCH$NUMBER OR TWO$CHANNELS;
 62   2                IF ERR$FLAG=1 THEN GOTO AGAIN1;
 64   2               AGAIN2:  CALL SEND$MESSAGE(.MS4);
 65   2                ENTRY$FLAG=1;
 66   2                    BM$PAGE=FETCH$NUMBER
 67   2                IF ERR$FLAG=1 THEN GOTO AGAIN2;
 69   2               REPEAT:  CALL SEND$MESSAGE(.MS9);
 70   2                    KEY=CI AND 7FH;
 71   2                CALL ECHO(KEY);
 72   2                CALL CROUT;
 73   2                IF KEY=YES THEN
 74   2                    DO;
 75   3                         CALL SEND$MESSAGE(.MS10);
 76   3                         RW$FLAG=1;
 77   3                    END;
                       ELSE
 78   2                    IF KEY<>NO THEN
 79   2                         DO;
 80   3                           CALL ILLEGAL$ENTRY;
 81   3                           GOTO REPEAT;
 82   3                         END;
 83   2     END IO$DEFINITION;
 84   1     END IOMOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 0360H    864D
     VARIABLE AREA SIZE = 0008H      8D
     MAXIMUM STACK SIZE = 0004H      4D
     154 LINES READ
     0 PROGRAM ERRORS
```

```
PL/M-80 COMPILER    MENU                                   6/6/83      PAGE 1


ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE MENU
OBJECT MODULE PLACED IN :F1:MENU.OBJ
COMPILER INVOKED BY: PLM80 :F1:MENU.PLM DEBUG


              $TITLE('MENU')
              $DATE(6/6/83)
1             MENU:DO;
              $INCLUDE(:F1:MSGMOD.LIT)
            = $NOLIST
20    1       DECLARE SP LITERALLY '20H';
21    1       DECLARE P BYTE;
22    1       DECLARE Q ADDRESS;
23    1       DECLARE ERR$FLAG BYTE EXTERNAL;                   /*BMCOM*/
24    1       CO:PROCEDURE (CHAR) EXTERNAL;                     /*MONITOR*/
25    2            DECLARE CHAR BYTE;
26    2            END CO;
27    1       SEND$MESSAGE:PROCESURE (PTR) EXTERNAL;            /*TERMIO*/
28    2            DECLARE PTR ADDRESS;
29    2            END SEND$MESSAGE;
```

```
          $EJECT
 30   1   MOVE$CURSOR:PROCEDURE (I,J);
         /*****************************************************************
         DESCRIPTION:  SENDS THE CONSOLE THE REQUESTED NUMBER OF CARRIAGE
                       RETURNS, LINE FEEDS AND SPACES

              INPUTS:  NUMBER OF CARRIAGE RETURNS, SPACES AND LINE FEEDS
                       DESIRED

             OUTPUTS:  CR, SP AND LF TO TERMINAL

               CALLS:  CO

            DESTROYS:  A
         *****************************************************************/

 31   2        DECLARE (I,J) BYTE;
 32   2        IF I <>0 THEN
 33   2             DO P=0 TO I;
 34   3                  CALL CO(CR);
 35   3                  CALL CO(LF);
 36   3             END;
 37   2        IF J <>0 THEN
 38   2             DO Q=0 TO J;
 39   3                  CALL CO(SP);
 40   3             END;
 41   2   END MOVE$CURSOR;
```

```
          $EJECT
42    1   PRINT$MENU:PROCEDURE PUBLIC;
         /*******************************************************************
         DESCRIPTION:  DISPLAYS A LIST OF THE COMMANDS IN MENU FORM

              INPUTS:  NONE

             OUTPUTS:  NONE

               CALLS:  SEND$MESSAGE,MOVE$CURSOR

            DESTROYS:  NONE
         *******************************************************************/

43    2         CALL MOVE$CURSOR(5,8);
44    2         DO P=0 TO 10;
45    3             Q=P*26;
46    3             CALL SEND$MESSAGE(.MSG(Q));
47    3             CALL MOVE$CURSOR(0,8);
48    3         END;
49    2         CALL MOVE$CURSOR(4,0);
50    2         CALL SEND$MESSAGE(.MSO);
51    2     END PRINT$MENU;
52    1     END MENU;

MODULE INFORMATION:

     CODE AREA SIZE     = 0368H     872D
     VARIABLE AREA SIZE = 0005H       5D
     MAXIMUM STACK SIZE = 0004H       4D
     120 LINES READ
     0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1          PARMET        PAGE 2
PARMET-1/13/83

LOC OBJ         LINE      SOURCE STATEMENT

                 37  PUBLIC SETPAR
                 38
                 39; ******************************************************************
                     **
                 40;
                 41; DESCRIPTION:  THIS PROGRAM SETS THE INITIAL VALUES OF THE
                 42;               PARAMETRIC REGISTERS OF THE 7220 BUBBLE MEMORY
                 43;               CONTROLLER
                 44;
                 45;      INPUTS:  B/C REGISTER-# OF PAGES TO BE TRANSFERRED
                 46;               D/E REGISTER-PAGE NUMBER IN BUBBLE MEMORY
                 47;
                 48;     OUTPUTS:  A REGISTER--RETURNS VALUE OF 7220 STATUS REGISTER
                 49;               MEM 3000-3005H--PARAMETRIC DATA
                 50;
                 51;       CALLS:  NONE
                 52;
                 53;    DESTROYS:  B,C,D,E,H,L,A
                 54;
                 55; ******************************************************************
                     */
                 56;
0013 210073      57 SETPAR:    LXI H, 7300H   ;BEGIN LOADING PARAMETRIC REGISTER DATA
0016 36FF        58            MVI M,OFFH     ;UTILITY REGISTER
0018 23          59            INX H          ;NEXT LOCATION
0019 71          60            MOV M,C        ;BLOCK LENGTH REGISTER LSB
001A 23          61            INX H          ;NEXT LOCATION
00AB 70          62            MOV M,B        ;BLOCK LENGTH REGISTER MSB
001C 23          63            INX H          ;NEXT LOCATION
001D 3600        64            MVI M,00H      ;ENABLE REGISTER
001F 23          65            INX H          ;NEXT LOCATION
0020 73          66            MOV M,E        ;ADDRESS REGISTER LSB
0021 23          67            INX H          ;NEXT LOCATION
0022 72          68            MOV M,D        ;ADDRESS REGISTER MSB
0023 C9          69            RET
                 70;
                 71 $EJECT
```

```
PL/M-80 COMPILER   TERMIO-1/11/83                              PAGE 1

ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE TERMIO
OBJECT MODULE PLACED IN :F1:TERMIO.OBJ
COMPILER INVOKED BY:  PLM80 :F1:TERMIO.PLM DEBUG


           $TITLE('TERMIO--1/11/83')
 1         TERMIO:DO;
 2    1    CI:PROCEDURE BYTE EXTERNAL;     /*ENTRY POINT INTO SYSTEM.LIB*/
 3    2         END CI;
 4    1    CO:PROCEDURE (CHAR) EXTERNAL;   /*ENTRY POINT TO SYSTEM.LIB*/
 5    2         DECLARE CHAR BYTE;
 6    2         END CO;
 7    1    ECHO:PROCEDURE (CHAR) BYTE;
 8    2         DECLARE CHAR BYTE;
 9    2         END ECHO;
10    1    DECLARE BUFSIZE LITERALLY '256';
11    1    DECLARE BUFFER (BUFSIZE) BYTE PUBLIC;
12    1    DECLARE INDEX BYTE PUBLIC;     /*INDEX INTO BUFFER*/
13    1    DECLARE CR LITERALLY '0DH';    /*CARRIAGE RETURN*/
14    1    DECLARE LF LITERALLY '0AH';          /*LINE FEED*/
15    1    READ$CONSOLE:PROCEDURE PUBLIC;
          /******************************************************************

          DESCRIPTION:  READS A STRING OF CHARACTERS FROM THE CONSOLE
                        DEVICE

               INPUTS:  CHARACTERS FROM THE CONSOLE

              OUTPUTS:  CHARACTER TO BUFFER AND TO CONSOLE

                CALLS:  CI;ECHO

             DESTROYS:  NONE

          ******************************************************************/

16    2         INDEX=0;
17    2         BUFFER(INDEX)=CI AND 7FH;   /*READ CHAR AND STRIP OFF PARITY*/
18    2         CALL ECHO(BUFFER(INDEX));
19    2         DO WHILE BUFFER(INDEX)<>CR;
20    3              IF INDEX<LAST(BUFFER) THEN
21    3                   DO;               /*CONTINUE READING UNTIL A*/
22    4                      INDEX=INDEX+1; /*CARRIAGE RETURN HAS BEEN*/
23    4                      BUFFER(INDEX)=CI AND 7FH; /*INPUT OR BUFFER IS
                                                          FULL*/
24    4                      CALL ECHO(BUFFER(INDEX));
25    4                            END;
26    3         END;
27    2              END READ$CONSOLE;
```

```
           $EJECT
28    1    SEND$MESSAGE:PROCEDURE (PTR) PUBLIC;
          /*****************************************************************

          DESCRIPTION:  OUTPUTS A STRING OF CHARACTERS TO THE CONSOLE

               INPUTS:  CHARACTER TO BE SENT

              OUTPUTS:  CHARACTER TO THE CONSOLE

                CALLS:  CO

             DESTROYS:  NONE

          *****************************************************************/

29    2         DECLARE PTR ADDRESS,CHAR BASED PTR(1) BYTE;
30    2              INDEX=0;
31    2              CALL CO(CHAR(INDEX));         /*OUTPUT FIRST CHARACTER*/
32    2              DO WHILE CHAR(INDEX)<> CR;    /*CONTINUE OUTPUTTING*/
33    3              INDEX=INDEX+1;           /*UNTIL A CARRIAGE RETURN*/
34    3                   CALL CO(CHAR(INDEX));    /*IS OUTPUT*/
35    3         END;
36    2              CALL CO(LF);
37    2    END SEND$MESSAGE;
```

```
          $EJECT
38   1    QUICK$CI:PROCEDURE BYTE PUBLIC;
          /*****************************************************************

         DESCRIPTION:  READS A CHARACTER FROM THE CONSOLE IF THERE IS ONE
                       RETURNS WITH A 00 IF NO CHARACTER IS READ.

              INPUTS:  CHARACTER FROM THE CONSOLE

             OUTPUTS:  CHARACTER READ

               CALLS:  NONE

            DESTROYS:  A

          *****************************************************************/

39   2    DECLARE X BYTE;
40   2    DECLARE READY LITERALLY '02H';
41   2    DECLARE CNINPT LITERALLY '0ECH';
42   2    DECALRE CNSTATPT LITERALLY '0EDH';
43   2         IF (INPUT (CNSTATPT) AND READY)<>READY THEN
44   2              X=0;
               ELSE
45   2              X=INPUT (CNINPT) AND 7FH;
46   2         RETURN X;
47   2    END QUICK$CI;
48   1    END TERMIO;


MODULE INFORMATION:

     CODE AREA SIZE     = 00BCH     188D
     VARIABLE AREA SIZE = 0104H     260D
     MAXIMUM STACK SIZE = 0002H       2D
     96 LINES READ
     0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER    TSTMOD--1/5/83                                PAGE 1

ISIS-II PL/M-80 V4.0 COMPILATION OF MODULE TSTMOD
OBJECT MODULE PLACED IN :F1:TDTMOD.OBJ
COMPILER INVOKED BY:  PLM80 :F1:TSTMOD.PLM DEBUG

              $TITLE('TSTMOD--1/5/83')
  1           TSTMOD:DO;
  2    1      CO:PROCEDURE (CHAR) EXTERNAL;            /*MONITOR*/
  3    2          DECLARE CHAR BYTE;
  4    2          END CO;
  5    1      CROUT:PROCEDURE EXTERNAL;                /*MONITOR*/
  6    2          END CROUT;
  7    1      SEND$MESSAGE:PROCEDURE (PTR) EXTERNAL; /*TERMIO*/
  8    2          DECLARE PTR ADDRESS;
  9    2          END SEND$MESSAGE;
 10    1      DECLARE CMD$NUMBER BYTE EXTERNAL;        /*BMCOM*/
 11    1      DECLARE C BYTE PUBLIC;
              $INCLUDE(:F1:MSGMOD.LIT)
            = $NOLIST
 30           TEST:PROCEDURE PUBLIC;
              /*************************************************************
             DESCRIPTION:  SENDS A MESSAGE TO THE SCREEN TO INDICATE WHICH
                           COMMAND IS EXECUTING

                  INPUTS:  NONE

                 OUTPUTS:  MESSAGE TO CRT

                   CALLS:  SEND$MESSAGE,CO,CROUT

                DESTROYS:  NONE

              *************************************************************/

 31    2          DO;
 32    3              C=CMD$NUMBER+30H;
 33    3              CALL SEND$MESSAGE(.MS12);
 34    3              CALL CO(C);
 35    3              CALL CROUT;
 36    3          END;
 37    2      END TEST;
 38    1      END TSTMOD;


MODULE INFORMATION:

     CODE AREA SIZE     = 02C1H     705D
     VARIABLE AREA SIZE = 0001H       1D
     MAXIMUM STACK SIZE = 0002H       2D
     88 LINES READ
     0 PROGRAM ERRORS

END OF PL/M-80 COMPILATION
```

1. Report No. NASA TM-85832	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Investigation of Fast Initialization of Spacecraft Bubble Memory Systems		5. Report Date June 1984
		6. Performing Organization Code 506-58-13-03
7. Author(s) Karen T. Looney Charles D. Nichols Paul J. Hayes		8. Performing Organization Report No.
9. Performing Organization Name and Address Langley Research Center Hampton, VA 23665		10. Work Unit No.
		11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		13. Type of Report and Period Covered Technical Memorandum
		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Bubble domain technology offers significant improvement in reliability and functionality for spacecraft onboard memory applications. In considering potential memory systems organizations, minimization of power in high capacity bubble memory systems necessitates the activation of only the desired portions of the memory. In power strobing arbitrary memory segments, a capability of fast turn-on is required. Bubble device architectures, which provide redundant loop coding in the bubble devices, limit the initialization speed. Alternate initialization techniques have been investigated to overcome this design limitation. An initialization technique using a small amount of external storage has been demonstrated. This technique provides several orders of magnitude improvement over the normal initialization time.

17. Key Words (Suggested by Author(s)) Bubble memory, Spacecraft memory system, Fast initialization		18. Distribution Statement Unclassified - Unlimited Subject Category 33	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 70	22. Price A04

For sale by the National Technical Information Service, Springfield, Virginia 22161